

# AJaX

Cyril Rabat

`cyril.rabat@univ-reims.fr`

Licence 2 Informatique - Info0303 - Programmation Web 2

2023-2024



**Cours n°5**

*Présentation d'AJaX*

*Version 15 septembre 2023*

# Table des matières

## 1 Présentation d'AJaX

- Rappels sur *Javascript*
- AJaX

## 2 La classe XMLHttpRequest

- Méthodes de base
- Exemple 1 : *Chuck Norris facts*
- Exemple 2 : envoi de données

## 3 AJaX : JQuery et Fetch

- Utilisation d'AJaX en *JQuery*
- Exemple d'application
- Utilisation d'AJaX avec *Fetch*

## Rappels sur *Javascript*

- Créé en 1995, standardisé sous ECMAScript en 1997  
↪ *Javascript* est une implémentation d'ECMAScript
- Langage de script exécuté au sein d'une page Web
  - Intégré dans le code HTML
  - Interprété par le navigateur  
↪ Attention cependant aux incompatibilités entre navigateurs !
- Basé en partie sur la syntaxe du *Java*
- Objectif : rendre la page dynamique
- Une API complète
- Différents composants/éléments/langages liés :
  - AJaX
  - JSON

## Quelques mots sur le cache

- Rappels lors de l'accès à une URL :
  - Script principal chargé par le navigateur
  - Éléments secondaires (images, scripts, *etc.*) : utilisation du cache
- Scripts *Javascript* situés dans des fichiers séparés :
  - ↪ Évite la surcharge du code HTML
  - ↪ Libère de la bande passante (scripts non rechargés à chaque fois)
- Problèmes dans le cas de la programmation *Javascript* :
  - ↪ Modifications des scripts non prises en compte sur le client !
- Solutions :
  - Vider l'historique du navigateur
  - Inclure le *Javascript* dans le fichier HTML (**en mode développement**)
  - Configuration du serveur Web (empêche la mise en cache)
  - Manuellement avec un numéro de version (**en mode développement**) :
    - ↪ `<script src="script.js?v=<?php echo time(); ?>" ...`

# AJaX

- AJaX pour **A**synchronous **J**avascript and **X**ML
- Exploiter *Javascript* pour l'envoi de requêtes HTTP
- Les réponses peuvent ensuite être analysées :
  - ↪ Généralement au format XML ou JSON
  - ↪ Peut être de tout type (exemples : JSON, HTML, *etc.*)
- Utilisation de la classe *Javascript* : XMLHttpRequest
- Du côté du serveur : aucun changement !

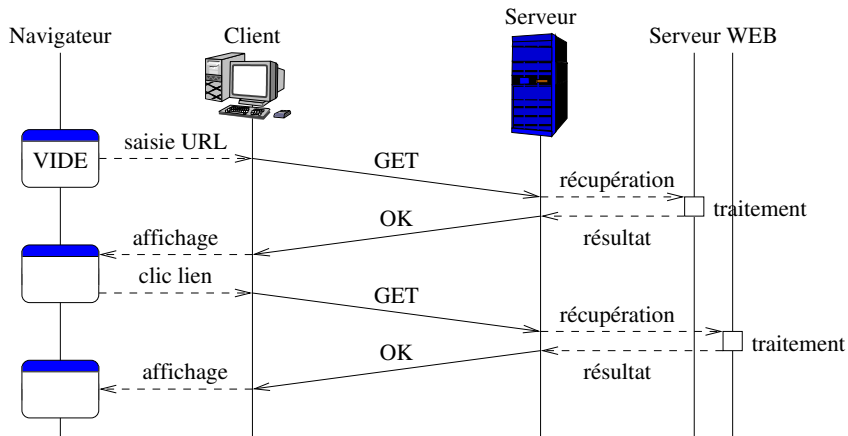
# Fonctionnement d'AJaX

- Sans AJaX, à chaque requête HTTP, nouvelle page créée par le navigateur
- Avec AJaX, possibilité d'exécuter des requêtes au sein d'une même page Web :
  - ↪ Pas de rechargement de la page
- Intérêts :
  - Pages dynamiques (modification du DOM de la page)
  - Plus grande ergonomie pour l'utilisateur
  - Pas de nouveau langage (réutilisation de standards)
  - Données présentes sur le client et sur le serveur
- Les appels peuvent être asynchrones...
  - ↪ L'utilisateur garde la main pendant la réception des données
- ...ou synchrones (mais obsolètes...)

# Limites d'AJaX

- Croisement de domaine :
  - Par mesure de sécurité, impossible de récupérer des données sur un autre domaine avec AJaX
  - Travail actuel du W3C, même si des implémentations (propriétaires) existent
- Techniquement, toutes requêtes HTTP autorisées
  - ↪ dont GET et POST
- Attention à l'encodage des données (UTF-8)

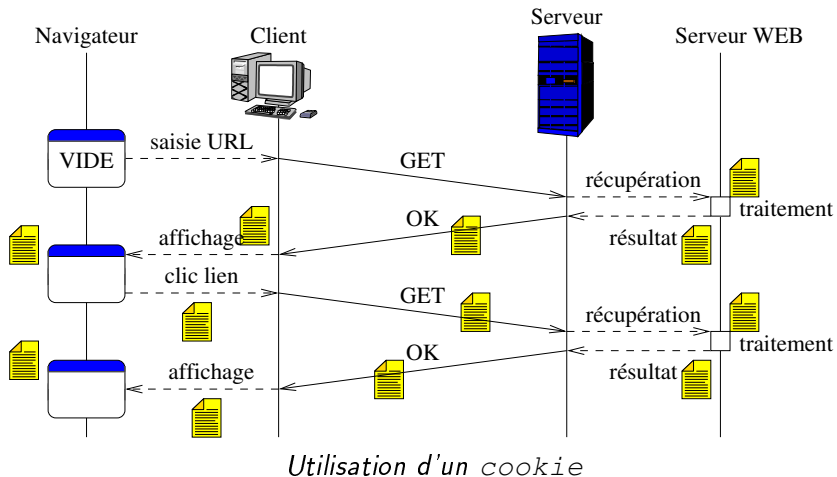
# Différences d'appels



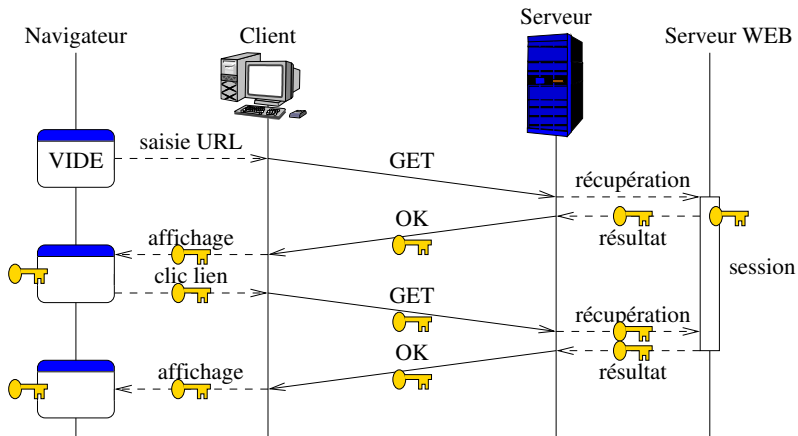
*Série de requêtes/réponses HTTP classiques*



# Différences d'appels



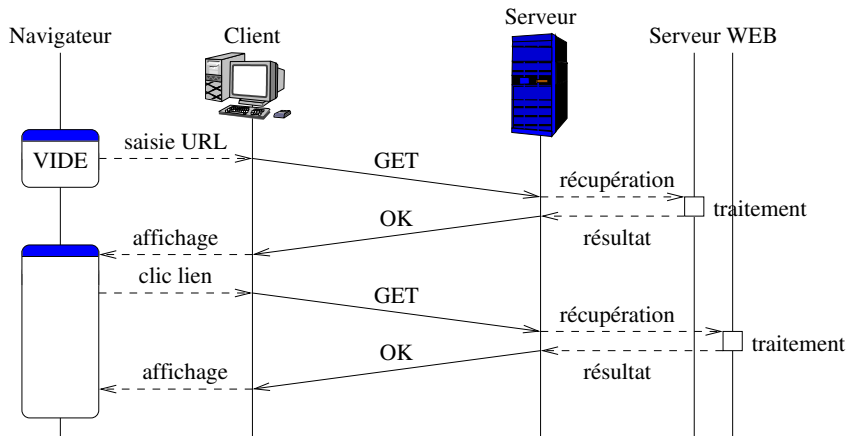
# Différences d'appels



*Utilisation d'une session*



# Différences d'appels



*Utilisation d'AJaX (couplage possible avec les cookies et les sessions)*

# Présentation de la classe XMLHttpRequest

- Manipulation d'AJaX :
  - ↪ Utilisation de la classe XMLHttpRequest
- Possède un constructeur par défaut :
  - ↪ `var requeteHTTP = new XMLHttpRequest();`
- Permet l'envoi de requêtes HTTP et le traitement de la réponse
- Si plusieurs requêtes simultanées, plusieurs objets nécessaires
- Une fois créé, il faut initialiser l'objet :
  - Préparation de la requête
  - Modification de l'en-tête

## Préparation de la requête et envoi

- Création de l'URL fourni à l'objet XMLHttpRequest :  
↪ `requeteHTTP.open("GET", "toto.html");`
- Le prototype de la méthode `open` :  
`open(méthode, URL, synchrone, user, password)`
  - `méthode` : GET, POST, HEAD, PUT, DELETE
  - `URL` : URL
  - `synchrone` (option) : `true` si appel asynchrone (par défaut)
  - `user` et `password` (option) : authentification HTTP
- Une fois l'URL spécifié, envoi de la requête :  
↪ `requeteHTTP.send();`

## Appel asynchrone

- Appel à `send` non bloquant
- À la réception de la réponse : un *handler* (ou *callback*) est appelé
- Plusieurs types de *handler* suivant l'état de l'appel :
  - `onloadstart`, `onload`, `onloadend`
  - `onprogress`
  - `onabort`
  - `onerror`
  - `ontimeout`
- Pour spécifier le *handler* :
  - ↪ `requeteHTTP.onloadend = function() { ... };`
- Il s'agit d'un pointeur de fonction : possible d'utiliser une fonction existante
  - ↪ `requeteHTTP.onloadend = traitement;`
  - ↪ La fonction `traitement` est définie ailleurs

## Handler générique

- Type de *handler* générique : `onreadystatechange`  
↪ Fonction associée appelée à chaque changement d'état de l'objet
- Le *handler* est appelé plusieurs fois!!!
- Comment savoir si la réponse a été reçue?  
↪ Vérification de l'état de l'objet

## L'état de l'objet

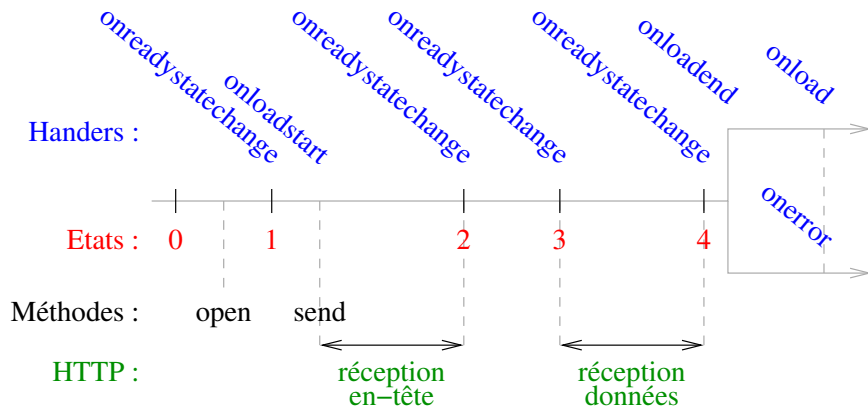
- L'objet XMLHttpRequest prend différents états successifs
- État courant récupéré à l'aide de l'attribut `readyState`
- Les différents états :
  - UNSENT (=0) : l'objet est construit
  - OPENED (=1) : la méthode `open` a été correctement invoquée
  - HEADER\_RECEIVED (=2) : en-tête final reçu
  - LOADING (=3) : corps en cours de réception
  - DONE (=4) : toutes les données ont été reçues



# Récupération du statut HTTP

- Deux attributs :
  - `status` : code d'état HTTP (exemple : 200)
  - `statusText` : message associé (exemple : OK)
- Utilisable dès que l'en-tête est reçu
- Par défaut à 0 (ou en cas d'erreur)

## Résumé d'un appel et états correspondants



## Description de l'exemple

- Page Web complétée dynamiquement :
  - Nouveaux éléments reçus via AJAX
  - Insertion dans le DOM
- Scripts utilisés :
  - Fichier HTML :
    - Contient le *Javascript* exploitant AJAX
    - Paragraphe vide complété avec la réponse AJAX
      - ↪ Identifiant *destAJAX*
      - ↪ Possible aussi de le construire à l'exécution
    - Bouton appelant la méthode qui lance la requête AJAX
  - Script PHP : génération de texte

## Code Javascript

```
<html>
<head>
  <script type="text/javascript">
    var requeteHTTP = new XMLHttpRequest();
    requeteHTTP.onload = function() {
      document.getElementById("destAJaX").innerHTML = requeteHTTP.responseText;
    }
    requeteHTTP.onerror = function() {
      document.getElementById("destAJaX").innerHTML = "Erreur_!";
    }
    function exemple() {
      requeteHTTP.open("GET", "generateur.php");
      requeteHTTP.send();
    }
  </script>
</head>
...
```

# Code PHP

```
<?php
define("CITATIONS", [
    "Peter_Parker_a_été_mordu_par_une_araignée,_Clark_Kent_a_été_mordu_par_Chuck_Norris.",
    ,
    "Chuck_Norris_se_souvient_très_bien_de_son_futur",
    "Chuck_Norris_a_déjà_compté_jusqu'à_l'infini._Deux_fois.",
    "Google,_c'est_le_seul_endroit_où_tu_peux_taper_Chuck_Norris...",
    "Chuck_Norris_donne_fréquemment_du_sang_à_la_Croix-Rouge._Mais_jamais_le_sien.",
    "Chuck_Norris_et_Superman_ont_fait_un_bras_de_fer,_le_perdant_devait_mettre_son_slip_
        par_dessus_son_pantalon.",
    "Les_laxatifs_sont_sans_effets_sur_Chuck_Norris,_on_ne_fait_pas_****_Chuck_Norris",
    "Chuck_Norris_peut_remonter_le_terrain_d'Olive_et_Tom_en_moins_d'un_épisode"
]);
echo CITATIONS[rand(0, count(CITATIONS) - 1)];
```

*Olive et Tom est anim japonais tiré du manga Captain Tsubasa  
Aucun stade climatisé n'a été utilisé lors des tournages*

## Envoi de données - Méthode GET

- Reprise de l'exemple précédent : le client choisit maintenant la citation
- Solution retenue : envoi d'une valeur par la méthode GET
- Les données sont codées dans l'URL :
  - ↪ Ajout de "?" puis des couples séparés par "&"
  - ↪ Chaque couple : `nomVariable=valeur`
- Utilisation de la fonction *Javascript* `encodeURIComponent` :
  - ↪ Encodage pour que l'URL soit valide
  - ↪ Non obligatoire pour un entier

# Fichier HTML

```
<html lang="fr">
<head>
  <script type="text/javascript">
    function exemple(numero) {
      var requeteHTTP = new XMLHttpRequest();
      requeteHTTP.onload = function() {
        document.getElementById("destAJaX").innerHTML = requeteHTTP.responseText;
      }
      requeteHTTP.open("GET", "generateur.php?numero=" + numero);
      requeteHTTP.send();
    }
  </script>
</head>
<body>
  <h1> Citation du jour </h1>
  <p id="destAJaX"></p>
  <button type="button" onclick="exemple(0)"> Citation 1 </button>
  <button type="button" onclick="exemple(1)"> Citation 2 </button>
  ...

```

# Script PHP

```
...  
// Écriture de la case choisie (ou aléatoire si non spécifiée)  
if(isset($_GET['numero']))  
    $numero = intval($_GET['numero']) % count(CITATIONS);  
else  
    $numero = rand(0, count(CITATIONS) - 1);  
  
echo CITATIONS[$numero];
```



## Envoi de données - Méthode POST

- Données non spécifiées dans l'URL
- Modifier l'en-tête avec le Content-Type suivant :
  - ↪ Pour les formulaires : `application/x-www-form-urlencoded`
- Données passées dans la méthode `send`
  - ↪ Attention à l'encodage !

```
function exemple(numero) {  
  var requeteHTTP = new XMLHttpRequest();  
  requeteHTTP.onload = function() {  
    document.getElementById("destAJaX").innerHTML = requeteHTTP.responseText;  
  }  
  requeteHTTP.open("POST", "generateur.php");  
  requeteHTTP.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");  
  requeteHTTP.send("numero=" + numero);  
}
```

# *JQuery*

- Bibliothèque *Javascript*
- Permet...
  - ...d'enrichir les possibilités de *Javascript*
  - ...de simplifier le développement
  - ...d'ajouter un système de sélecteurs puissant
  - ...d'ajouter des effets (plus beaucoup utilisé grâce aux fonctionnalités du CSS)
  - *etc.*
- À inclure comme un script *Javascript* :
  - ↪ Utiliser le CDN

## AJaX avec *JQuery*

- Solution permettant de simplifier les appels AJaX
- Fonction `ajax` qui prend en paramètres :
  - Le type de requête HTTP (GET, POST, *etc.*)
  - L'URL
  - Les données
    - ↪ Format standard (`"nom=" + nom + "&prenom=" + prenom`)
    - ↪ JSON (`{ 'nom' : nom, 'prenom' : prenom }`)
  - Les *handlers* pour la réussite, l'échec
- Permet aussi simplement de modifier l'en-tête :
  - Type des données envoyées
    - ↪ Exemple : `application/x-www-form-urlencoded`
  - Type des données à recevoir
  - *etc.*
- Objet `jqXHR` retourné

## Code générique d'une requête AJaX

```
requete = $.ajax(  
  {  
    type: 'POST',  
    url: '',  
    data: '',  
    dataType: 'json'  
  }  
);  
requete.done(  
  function (reponse, texteStatut, jqXHR) {  
    // Requête réussie  
  }  
);  
requete.fail(  
  function (jqXHR, texteStatut, erreurLevee){  
    // Erreur !!!  
  }  
);
```

## Exemple : les familles des séries télévisées

- Application permettant de récupérer les personnages de séries
- Constituée des fichiers suivants :
  - `index.html` : formulaire pour la saisie
  - `script.js` : script *Javascript* avec la requête AJaX
    - ↔ Interagit avec `donnees.php`
  - `series.json` : données brutes (format JSON)
  - `donnees.php` : retourne les données en fonction des saisies de l'utilisateur
- Fonctionnement :
  - L'utilisateur saisit le nom de la série et éventuellement, le nom de la famille
  - La liste des familles ou des membres est affichée

## Extrait du fichier index.html

```
...  
<label ref="serie">Série</label>  
<input type="text" id="serie"/>  
<label ref="famille">Famille</label>  
<input type="text" id="famille"/>  
  
<button type="button" onclick="recherche() ">  
  Recherche  
</button>  
  
<div id="erreur"></div>  
<div id="resultat"></div>  
  
<script src="https://codeorigin.jquery.com/jquery-3.7.1.min.js"></script>  
<script src="script.js" type="text/javascript"></script>  
...
```

## Le script PHP `donnees.php`

- Récupère le nom de la série et éventuellement, le nom de famille  
↪ En POST
- Chargement des données depuis le JSON
- Retourne un document JSON contenant :
  - Le code de retour (OK ou KO)
  - La liste des données (séries, familles, personnages)

## Extrait du fichier *Javascript* `script.js` (1/2)

```
function recherche() {  
  requete = $.ajax({  
    type: 'POST',  
    url: 'donnees.php',  
    data: { "serie" : $('#serie').val(),  
           "famille" : $('#famille').val() },  
    dataType: 'json'  
  });  
  requete.fail(function (jqXHR, textStatus, errorThrown){  
    $('#erreur').text("Problème_de_requête_AJaX.").show();  
    console.log(jqXHR);  
  });  
  ...  
}
```



## Extrait du fichier *Javascript* script.js (2/2)

```
...
requete.done(function (response, textStatus, jqXHR) {
    if("erreur" in response)
        $('#erreur').text(response['erreur']).show();
    else
        $('#erreur').hide();

    $('#resultat').text(response['titre']).append('<ul>').show();
    for(i = 0; i < response['donnees'].length; i++)
        $('#resultat').append("<li>" + response['donnees'][i] + "</li>");
    $('#resultat').append("</ul>");
});
}
```

## *Fetch*

- Alternative à XMLHttpRequest
- Définition générique des objets Request et Response
- Plus de fonctionnalités
- Apporte de la souplesse et de la puissance
- Pas besoin d'inclure *JQuery*

Comme toute API, toutes les fonctionnalités ne sont pas supportées par tous les navigateurs

# Requête AJaX

```
function recherche() {  
  var serie = document.getElementById('serie').value;  
  var famille = document.getElementById('famille').value;  
  fetch('donnees.php', {  
    method: "POST",  
    body: 'serie=' + encodeURIComponent(serie) +  
          '&famille=' + encodeURIComponent(famille),  
    headers: {  
      'Accept' : 'application/json',  
      'Content-Type': "application/x-www-form-urlencoded"  
    }  
  }).then(response => {  
    if(response.ok) {  
      response.json().then(analyse).catch(error => {  
        console.error(error);  
      });  
    } else  
      console.error(response.status);  
  }).catch(error => { console.error(error); });  
}
```

...

## Traitement de la réponse

```
function analyse(response) {
  var resultat = document.getElementById('resultat');
  var erreur = document.getElementById('erreur')

  if("erreur" in response) {
    erreur.innerHTML = response['erreur'];
    erreur.style.display = "block";
  }
  else
    erreur.style.display = "none";

  resultat.style.display = "block";
  resultat.innerHTML = response['titre'] + "<ul>";
  for(i = 0; i < response['donnees'].length; i++)
    resultat.innerHTML += "<li>" + response['donnees'][i] + "</li>";
  resultat.innerHTML += "</ul>";
}
```