

# 06.HTTP & AJAX

27 novembre 2023

## Développement web il3

### HTTP & AJAX

HE-Arc (DGR) 2022

### HyperText Transfer Protocol

- Protocole application : invention www en 1990 (v0.9)
  - Connexion, GET, réponse, fermeture
- HTTP 1.0 (1996)
  - Entêtes de requête (Host, Referer, User-Agent, ...) et réponse (Content-Type, Set-Cookie, Location, ...)
- HTTP 1.1 (1997)
  - Nouveaux entêtes (Keep-alive, pipelining, cache, ...), Host obligatoire
- HTTP 2.0<sup>1</sup> (2015)
  - Binaire, multiplexage connexions, compression entêtes, push, ...
  - Supporté par presque tous<sup>2</sup> les navigateurs, une majorité de serveurs
- HTTP 3.0<sup>3</sup> (2019)
  - UDP, correction erreur, contrôle congestion, multiplexage (0 RTT)

---

<sup>1</sup><https://docs.google.com/presentation/d/1eqae3OBCxwWswOsaWMAWRpqnmrVVrAfPQclfSqPkXrA/present#slide=id.p19>

<sup>2</sup><https://caniuse.com/#feat=http2>

<sup>3</sup><https://http3-explained.haxx.se/fr/>

## Codes de réponse

- 1xx : Information
- 2xx : Succès
- 3xx : Redirection
- 4xx : Erreur Client
- 5xx : Erreur Serveur

## Méthodes HTTP (verbes)

- GET : Demander une ressource
- POST : Création d'une ressource
- PUT : Remplacement total d'une ressource
- PATCH : Remplacement partiel d'une ressource
- DELETE : Suppression d'une ressource
- HEAD : Demande l'entête de la réponse, sans la ressource
- TRACE, OPTIONS, CONNECT

idempotentes sûres

## Echanges HTTP

- Requête

```
GET / HTTP/1.1[CRLF]
Host: www.cff.ch[CRLF]
Connection: close[CRLF]
User-Agent: Opera/9.20 (Windows NT 6.0; U; en)[CRLF]
Accept-Encoding: gzip[CRLF]
Accept-Charset: ISO-8859-1,UTF-8;q=0.7,*;q=0.7[CRLF]
Cache-Control: no[CRLF]
Accept-Language: de,en;q=0.7,en-us;q=0.3[CRLF]
Referer: http://web-sniffer.net/[CRLF]
[CRLF]
```

- Réponse

```
HTTP Status Code: HTTP/1.1 302 Found
Date: Mon, 16 Nov 2009 08:01:35 GMT
Server: Apache
Location: http://www.sbb.ch/fr/
```

Content-Length: 205  
Connection: close  
Content-Type: text/html; charset=iso-8859-1

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head><title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="http://www.sbb.ch/fr/">here</a>.</p>
</body></html>
```

## HTTP

- Requête POST : paramètres dans le corps

```
POST /login.jsp HTTP/1.1
Host: www.mysite.com
User-Agent: Mozilla/4.0
Content-Length: 27
Content-Type: application/x-www-form-urlencoded
```

userid=joe&password=guessme

- Outils HTTP
  - CLI : curl
  - Browser dev tools
- Exemples PATCH : mnot<sup>4</sup>, SOA bits<sup>5</sup>

## AJAX : Historique

- Asynchronous Javascript And Xml
- Buzzword, Jesse James Garret<sup>6</sup>, 2005
- Mise à jour sans rechargement intégral
- Utilisation de Remote Scripting<sup>7</sup> et de DOM
- Historique de techniques de remote scripting

---

<sup>4</sup><https://www.mnot.net/blog/2012/09/05/patch>

<sup>5</sup><https://soabits.blogspot.ch/2013/01/http-put-patch-or-post-partial-updates.html>

<sup>6</sup><https://web.archive.org/web/20110102130434/http://www.adaptivepath.com/ideas/essays/archives/000385.php>

<sup>7</sup>[https://en.wikipedia.org/wiki/Remote\\_scripting](https://en.wikipedia.org/wiki/Remote_scripting)

- (i)frames
  - Bibliothèques JS (ex: JSRS<sup>8</sup>)
  - Utilisation des images/cookies (ex: GIF<sup>9</sup>)
  - Applets, Flash, ActiveX, ...
  - XHR : XML HTTP Request (IE5, 1999 pour OWA)
  - Fetch API
- Pas obligatoire d'avoir du JS, XML ni d'être asynchrone !

## AJAX

- XHR est devenue la méthode standard jusqu'à 2018
  - Popularisée par Google (GMaps, GMail, ...)
  - Le w3c fait évoluer un draft<sup>10</sup> depuis 2006
- Principe
  1. Envoi de requête HTTP
  2. La réponse provoque l'exécution de la fonction de rappel
  3. Le DOM de la page est mis à jour
- Applications
  - GUI ressemblant à des app natives
  - MAJ dynamiques de formulaires, autocompletion
  - Validation avec interrogation du serveur
  - ...

## L'objet *XMLHttpRequest*

- Initiative de Microsoft
  - Composant ActiveX de IE5
  - Adopté par Mozilla 1.0 et Safari 1.2
  - Standardisation W3C en cours
- Requête HTTP en JS
- Fonction de rappel (callback)
- Asynchrone : Non bloquant
- Non standard => différentes implémentations

---

<sup>8</sup><https://www.ashleyit.com/rs/jsrs/test.htm>

<sup>9</sup><https://web.archive.org/web/20100916110710/http://depressedpress.com/Content/Development/JavaScript/Articles/GIFAsPipe/Index.cfm>

<sup>10</sup><https://www.w3.org/TR/XMLHttpRequest/>

- Supporté par la majorité des navigateurs
- Alternative souhaitable si JS désactivé

## XHR en JS

```
var xhr;
function createXMLHttpRequest()
{
    if (window.ActiveXObject)
    {
        xhr = new ActiveXObject("Microsoft.XMLHTTP");
    }
    else if (window.XMLHttpRequest)
    {
        xhr = new XMLHttpRequest();
    }
}
```

- Dans son contexte<sup>11</sup>

## XHR en jQuery avec *load()*

```
<!DOCTYPE html>
<html>
<head>
<script src="jquery.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("#div1").load("demo_test.txt");
    });
});
</script>
</head>

<body>
    <div id="div1"><h2>Let jQuery AJAX Change This Text</h2></div>
    <button>Get External Content</button>
</body>
</html>
```

---

<sup>11</sup><https://www.xul.fr/xml-ajax.html#ajax-exemple>

- Tester<sup>12</sup>
- D'autres<sup>13</sup> façons de faire

## XHR : propriétés et méthodes

- readyState, status, onreadystatechange
- .responseText, responseXML
- open (Verbe, URI, async) :
  - Verbe HTTP : "GET", "POST" ou "PUT"
  - URI : destinataire de la requête
  - async (bool) : true = asynchrone, false = bloquant
- send (null | string) : peut être bloquante
- setRequestHeader(header, value)
- getResponseHeader(string)
- abort()

## Envoi de données

- GET
  - Obtenir des données
  - Longueur URL limitée par le navigateur (2'048 pour IE)
  - Utilise le cache (navigateur, proxy)
  - manipulables par l'utilisateur (bookmarks, partage, ...)
- POST
  - Faire quelque chose
  - Données sensibles
  - Longueur limitée par le serveur (assez large)
  - Utilisation de la méthode send() de XHR
  - Requête Ajax en 2 temps (entête, puis données)

## Envoi de données

- Cache

---

<sup>12</sup>[https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery\\_ajax\\_load](https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_ajax_load)

<sup>13</sup><https://code.tutsplus.com/tutorials/jquery-succinctly-jquery-and-ajax--net-33856>

- Client : Construire des URL uniques<sup>14</sup>
- Serveur : Envoi d'entêtes<sup>15</sup> interdisant le cache

```
MyXhr.open("GET", "fichier.xml", true);
MyXhr.setRequestHeader("Cache-Control", "no-store, no-cache, must-revalidate,
    post-check=0, pre-check=0");
MyXhr.setRequestHeader("Pragma", "no-cache");
MyXhr.setRequestHeader("Expires", "Wed, 09 Aug 2000 08:21:57 GMT");
```

## Préférer GET, sauf

Détails<sup>16</sup>

## Réponse en texte

- Si la requête aboutit :
  - `readystate == 4`
  - `status == 200`
- La réponse est dans l'attribut `responseText`
- ou dans `responseXML`
  - Utilisation du DOM (`getElementsByTagName()`, ...)

## Réponse en XML

```
<?xml version="1.0" ?>
<liste>
  <personne>
    <nom>Berger</nom>
    <prenom>Laurent</prenom>
  </personne>
  <personne>
    <nom>Borgo</nom>
    <prenom>Sébastien</prenom>
  </personne>
  <personne>
```

<sup>14</sup><https://stackoverflow.com/questions/367786/prevent-browser-caching-of-jquery-ajax-call-result>

<sup>15</sup><https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching>

<sup>16</sup><https://blog.teamtreehouse.com/the-definitive-guide-to-get-vs-post>



Figure 1: "GETorPOST"



```

        <nom>Bux</nom>
        <prenom>Rémy</prenom>
    </personne>
</liste>

```

- Dans responseXML

## Réponse en JSON<sup>17</sup>

- Standard<sup>18</sup> depuis octobre 2013 (Douglas Crockford<sup>19</sup>)
- Tableau d'objets js :
  - pour chacun, ses attributs sont des paires clé:valeur

```
{ "nom": "Berger", "prenom": "Laurent" }
```

```
[ "zéro", 1, 2, 3 ]
```

```
[
  { "nom": "Berger", "prenom": "Laurent" },
  { "nom": "Borgo", "prenom": "Sébastien" },
  { "nom": "Bux", "prenom": "Rémy" }
]
```

- Utilisation de : `var users = eval('(' + myXHR.responseText + ')');` pour créer le tableau d'objets correspondant

## « eval is Evil »<sup>20</sup>

- `eval()` : évalue et exécute la chaîne en paramètre
- Risque : instructions au lieu d'un tableau d'objets
- Solution : le parser<sup>21</sup> JSON

```

var users = JSON.parse(myXHR.responseText);
var myString = JSON.stringify(users);

```

- Avec jQuery :

---

<sup>17</sup><https://www.json.org/>

<sup>18</sup><https://ecma-international.org/publications-and-standards/standards/ecma-404/>

<sup>19</sup><https://www.crockford.com/>

<sup>20</sup><https://javascriptweblog.wordpress.com/2010/04/19/how-evil-is-eval/>

<sup>21</sup>[https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets\\_globaux/JSON/parse](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/JSON/parse)

```
var obj = jQuery.parseJSON('{ "nom": "Berger" }');  
alert(obj.nom);
```

## Fetch API

- Le successeur d'XHR est fetch<sup>22</sup> : Exemple<sup>23</sup>
- Fetch a un *polyfill* pour les navigateurs ne le supportant pas
- L'API Fetch est native et plus simple d'utilisation que jQuery

```
fetch("fichier.json")  
  .then(function(response) {  
    return response.json()  
  })  
  .then(function(json) {  
    console.log(json);  
  })  
  .catch(function(error) {  
    console.error("erreur", error)  
  })
```

- L'API fetch est native et utilise les promesses<sup>24</sup> plutôt que les callbacks

## Traitement d'erreurs

- Utiliser les entêtes HTTP<sup>25</sup>
  - Champ Status
  - Code d'erreur
- En PHP

```
header("Status: Message d'erreur explicite", true, 400);
```

- Afficher le message au client :

```
myXHR.getResponseHeader("Status");
```

---

<sup>22</sup><https://fetch.spec.whatwg.org/>

<sup>23</sup>[https://developer.mozilla.org/fr/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch)

<sup>24</sup><https://www.promisejs.org/>

<sup>25</sup><https://www.bennadel.com/blog/1860-using-appropriate-status-codes-with-each-api-response.htm>

## Penser à l'utilisateur !

- Requêtes XHR non enregistrées dans l'historique :
  - Bouton précédent non opérationnel (sauf GET et URL uniques)
  - Pas de bookmark
  - solution via History API<sup>26</sup>
- Utilisabilité : signaler à l'utilisateur ce qui est en cours :
  - GIF AJAX loading<sup>27</sup>
  - Rectangle Loading en haut à droite (Google)
  - Yellow Fade Technique<sup>28</sup> (37signals) : partie modifiée
- Code client :
  - Pas de maîtrise performance
  - Mauvais code == Appli lente
- En cas de doute, faire tester des utilisateurs

## Bonnes pratiques d'utilisabilité

- Trafic minimal
- Pas de surprise
- Respect des conventions
- Pas de distraction
- A11y (Contrast Checker<sup>29</sup>, Checklist<sup>30</sup>, ARIA<sup>31</sup>, Resources<sup>32</sup>)
- Ne pas switcher AJAX/non-AJAX
- Se mettre à la place de l'utilisateur

## Sources

---

<sup>26</sup><https://html.spec.whatwg.org/multipage/nav-history-apis.html>

<sup>27</sup><https://loading.io/>

<sup>28</sup><https://codepen.io/Mestika/pen/KVGwKb>

<sup>29</sup><https://color.a11y.com/>

<sup>30</sup><https://www.a11yproject.com/checklist/>

<sup>31</sup><https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA>

<sup>32</sup><https://www.a11yproject.com/resources/>