

06.HTTP & AJAX

9 octobre 2018

Développement web dlm3

HTTP & AJAX

HE-Arc (DGR) 2017

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, ...)
 - Entêtes de réponse (Content-Type, Set-Cookie, Location, ...)
- HTTP 1.1 (1997)
 - Keep-alive, pipelining, cache, ...
 - Plus d'entêtes, Host obligatoire
- HTTP 2.0¹ (2015)
 - Binaire, multiplexage connexions, compresions entêtes, push, ...
 - Supporté par presque tous² les navigateurs, une majorité de serveurs

Codes de réponse

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

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

²<http://caniuse.com/#feat=http2>

- 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
 - WebApp : HURL³
- Exemples PATCH : mnot⁴, SOA bits⁵

AJAX : Historique

- Asynchronous Javascript And Xml
- Buzzword, Jesse James Garret⁶, 2005
- Mise à jour sans rechargement intégral
- Utilisation de Remote Scripting⁷ et de DOM
- Historique de techniques de remote scripting
 - (i)frames⁸
 - Bibliothèques JS (ex : JSRS⁹)
 - Utilisation des images/cookies (ex : GIF¹⁰)
 - Applets, Flash, ActiveX, ...
 - XHR : XML HTTP Request (IE5, 1999 pour OWA)
 - Fetch API

³<http://hurl.it/>

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

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

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

⁷https://en.wikipedia.org/wiki/Remote_scripting

⁸<http://archive.oreilly.com/pub/a/javascript/2002/02/08/iframe.html>

⁹<http://www.ashleyit.com/rs/jsrs/test.htm>

¹⁰<http://web.archive.org/web/20100916110710/http://depressedpress.com/Content/Development/JavaScript/Articles/GIFAsPipe/Index.cfm>

- Pas obligatoire d'avoir du JS, XML ni d'être asynchrone !

AJAX

- XHR est devenue la méthode standard
 - Popularisée par Google (GMaps, GMail, ...)
 - Le w3c fait évoluer un draft¹¹ 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
- Supporté par Chrome, FF, Safari, IE, Konqueror, ...
- Alternative souhaitable si JS désactivé

XHR en JS

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

¹¹<https://www.w3.org/TR/XMLHttpRequest/>

```

    }
    else if (window.XMLHttpRequest)
    {
        xhr = new XMLHttpRequest();
    }
}

```

- Dans son contexte¹²

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>

```

- Tester¹³
- D'autres¹⁴ façons de faire

XHR : propriétés et méthodes

- readyState, status, onreadystatechange
- responseText, responseXML
- open (Verbe, URI, async) :
 - Verbe HTTP : "GET", "POST" ou "PUT"

¹²<http://www.xul.fr/xml-ajax.html#ajax-exemple>

¹³http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_ajax_load

¹⁴<https://code.tutsplus.com/tutorials/jquery-succinctly-jquery-and-ajax--net-33856>

- 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)
- Cache
 - Client : Construire des URL uniques¹⁵
 - Serveur : Envoi d'entêtes¹⁶ 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

Détails¹⁷

Réponse en texte

- Si la requête aboutit :

¹⁵<http://stackoverflow.com/questions/367786/prevent-browser-caching-of-jquery-ajax-call-result>

¹⁶<https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching>

¹⁷<http://blog.teamtreehouse.com/the-definitive-guide-to-get-vs-post>

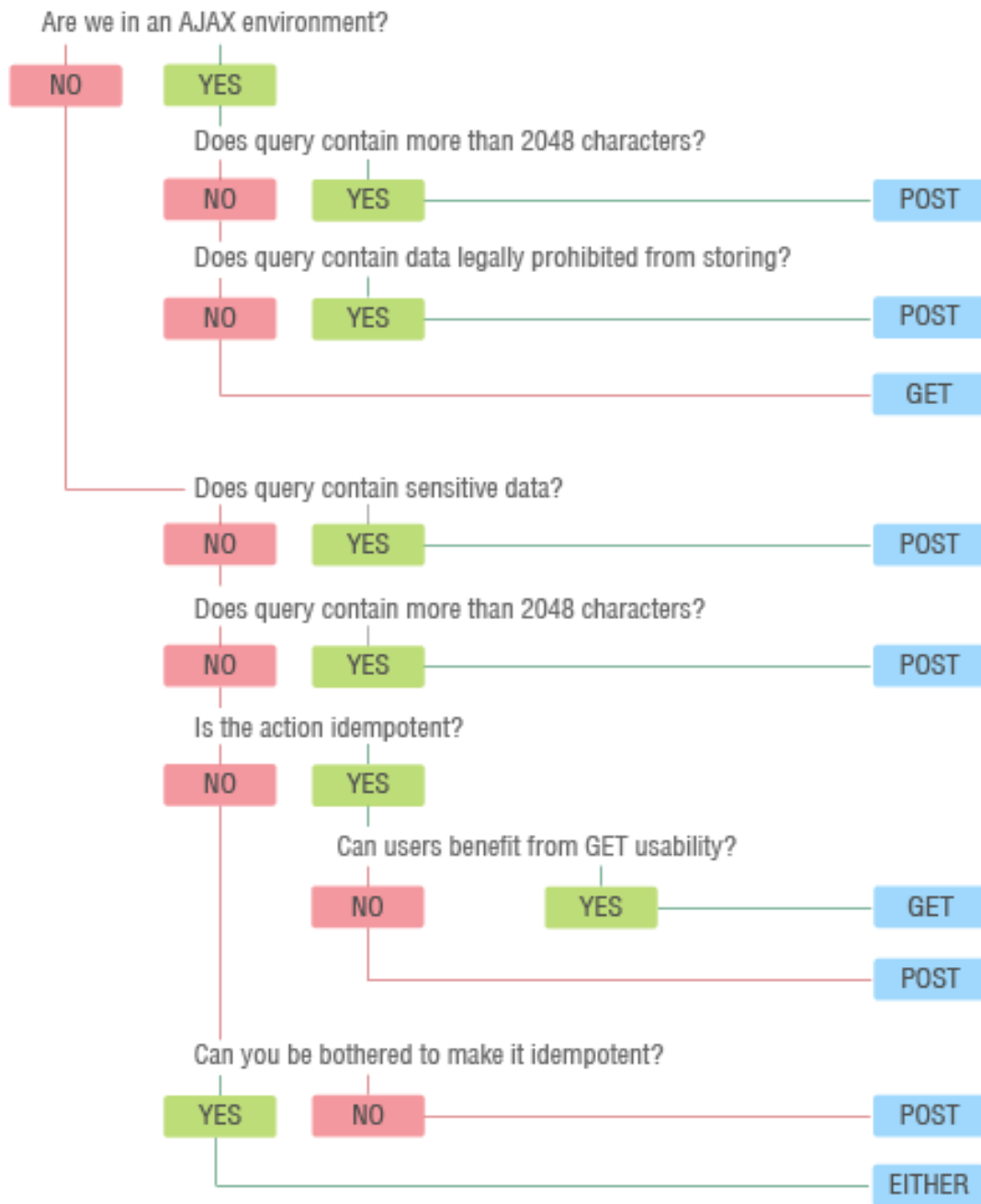


FIG. 1 : "GETorPOST"

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

- Dans responseXML

Réponse en JSON¹⁸

- Standard¹⁹ depuis octobre 2013 (Douglas Crockford²⁰)
- Tableau d'objets js :
 - pour chacun, ses attributs sont des paires clé : valeur

```
{objet1 : nom : 'Berger', prenom: 'Laurent'}
```

```
[objet1, objet2, objet3]
```

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

¹⁸<http://www.json.org/>

¹⁹<http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>

²⁰<http://www.crockford.com/>


```
    {nom: "Bux",      prenom: "Rémy"}  
  ]
```

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

« eval is Evil »²¹

- `eval()` : évalue et exécute la chaîne en paramètre
- Risque : instructions au lieu d'un tableau d'objets
- Solution : le parser²² JSON

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

- Avec jQuery :

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

Fetch API

- Le successeur d'XHR est fetch²³ : Exemple²⁴
- 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 promesses²⁵ plutôt que les callbacks

²¹<https://javascriptweblog.wordpress.com/2010/04/19/how-evil-is-eval/>

²²https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/JSON/parse

²³<https://fetch.spec.whatwg.org/>

²⁴https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch

²⁵<https://www.promisejs.org/>

Traitement d'erreurs

- Utiliser les entêtes HTTP²⁶
 - Champ Status
 - Code d'erreur
- En PHP

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

- Afficher le message au client :

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

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 API²⁷
- Utilisabilité : signaler à l'utilisateur ce qui est en cours :
 - GIF AJAX loading²⁸
 - Rectangle Loading en haut à droite (Google)
 - Yellow Fade Technique²⁹ (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
- Accessibilité (ARIA³⁰)
- Ne pas switcher AJAX/non-AJAX
- Se mettre à la place de l'utilisateur

²⁶<https://www.bennadel.com/blog/1860-using-appropriate-status-codes-with-each-api-response.htm>

²⁷<http://w3c.github.io/html/browsers.html#session-history-and-navigation>

²⁸<http://www.ajaxload.info/>

²⁹<https://signalnoise.com/archives/000558.php>

³⁰<https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA>

Sources