Ajax : accéder à des services web depuis le navigateur

le web classique

- **traitement classique d'une requête par le navigateur :**
  - envoi de la requête (GET ou POST)
  - attente bloquante du résultat
  - **remplacement** du document courant par le résultat, à son arrivée
- **conséquences :**
  - interface figée lors des échanges navigateur/serveur
  - rechargement complet de la page : html + css + js
  - tous les scripts js sont rechargés et ré-exécutés
  - le serveur doit générer une interface complète à chaque requête

les appels ajax

- un appel **"ajax"** : envoi **par programme** (js) d'une requête http (get, post ...) vers un serveur
- **la page n'est pas rechargée** : la requête retourne des données qui doivent être traitées par programme
- **fonctionnement asynchrone :** la **fonction** qui envoie la requête n'est pas la même que celle qui traitera la réponse
  - appel non bloquant : le **programme js continue son exécution,** l'interface reste active,
  - l'arrivée du résultat de la requête provoque un événement : les données sont transmises au handler correspondant

appel asynchrone

```
// appel synchrone
var v ;
v = grosCalcul( x );
console.log( v ) ;
```

EVT : arrivée des données

client : programme js

une fonction d'envoi

exécution

```
function send( handler ) { .. }
```

exécution

evt

handler(evt)

exécution

une (autre) fonction pour la réception

requête http

réponse http

serveur

les requêtes

- **Un appel ajax** : 1 requête HTTP
  - une url + 1 méthode HTTP (GET, POST ...)
  - des données à transmettre (dans l'url ou dans le corps)
  - un **callback** pour traiter le résultat en cas de succès : en général, décodage des données renvoyées + insertion dans le DOM
  - un **callback** pour traiter les cas de retour avec erreur

- **par défaut** : autorisé uniquement sur le domaine d'origine de la page
  - les requêtes ***cross-domain*** sont possibles si le serveur l'autorise en plaçant des headers particuliers dans la réponse(cors)

les réponses

- 1 **réponse** : 1 réponse HTTP
  - status : code de retour
  - des headers
  - un corps contenant des données
  - **les données sont transmises à la fonction callback**
- format des données :
  - texte simple : bof !
  - fragment html : à insérer directement dans l'arbre html – pratique à éviter : casse la séparation structure /présentation / comportement
  - données en format neutre : **xml** ou **json**, à traiter en js pour mettre à jour de dom

json : javascript objet notation

- Format évaluable en javascript
- objet :
  - { "a1" : v1 , "a2" : v2 …}
- tableau :
  - [v1, v2, v3 ..]
- valeur :
  - un objet, un tableau,
  - un littéral : "string", numérique
  - `null`, `true`, `false`

```
{
  "nom" : "marathon des neiges",
  "lieu": "Courcheneige",
  "classement": [
    21,345,42, 73
  ],
  "inscriptions": false,
  "img": null,
  "nombre_inscrits": 4273,
  "organisateur": {
    "nom": "michel",
    "mail": "michel@marathon-neiges.fr",
    "adresse": { "voie" : "rue de la glace",
                 "numero" : 32,
                 "code_postal" : "73502",
                 "ville": "Courcheneige"
    }
  }
}
```

ajax en javascript

- 1 objet de type **XmlHttpRequest** (xhr) :
  - des **méthodes** pour configurer et envoyer la requête et pour accéder à la réponse,
  - des **propriétés** pour accéder aux données retournées
    - **xhr.status** : le status de la réponse (200, 404, 500 …)
    - **xhr.responseText** : le body de la réponse, contient les données renvoyées par le serveur
    - **xhr.responseType** : le type de la réponse
  - des **événements** :
    - "`loadstart`" : le chargement démarre
    - "`progress`" : le chargement est en cours
    - "`error`" : le chargement à échoué
    - "load" : le chargement a réussi

```
function show_github_users(e) {
    "use strict"
    console.log(e);
    console.log(this);
    console.log(this.responseText);
}

let xhr = new XMLHttpRequest();
xhr.open('GET', 'https://api.github.com/users');

xhr.addEventListener("load", show_github_users);


xhr.send();
```

ajax avec jQuery

- jQuery propose une interface de haut niveau facilitant la mise en œuvre de requêtes ajax en se basant sur le principe des ***promises*** (promesses)

- les méthodes de création de requête ajax :
  - envoient la requête de façon asynchrone,
  - puis retournent 1 objet de type **jqXHR** qui implante l'interface ***Promise***
- l'interface ***Promise*** permet d'enregistrer les callbacks appelés
  - en cas de succès
  - en cas d'erreur

création et envoi d'une requête

- `jQuery.ajax( url, { settings } )` : appel non bloquant
- settings : objet de paramétrage
  - **type :** type de la requête - 'GET' ou 'POST'
  - **data** : { … } ou string – données transmises dans la requête (POST ou GET)
  - **context** : ce qui servira de this dans le callback appelé à la réception des données
  - **xhrFields** : un objet contenant des propriétés transmises à l'objet XHR – par exemple :

    `xhrFields : { withCredentials : true }` , pour transmettre les credentials dans 1 requête crossDomain
- retourne une ***promise*** sous la forme d'un objet **jqXHR**

promise et callback

- la promise permet d'enregistrer les callbacks qui seront appelés lors du retour de la réponse :

  - jqXHR.**done**(function(data, status, jqXHR) { }) : enregistrement d'un callback appelé en cas de succès
    - data : les données reçues,
    - status : statut de la requête ('success')
  - jqXHR.**fail**(function(jqXHR, status, error) { }) : enregistrement d'un callback appelé en cas d'erreur
    - status : "timeout", "error", "abort"
    - error : status http textuel → "Not Found", "Internal Server Error" …

```
<div class="vignette">
<img src="/photos/miniature/paysage.2.jpg" alt="">
<a href="/photos/paysages/2"class="img-lk"> voir +</a>
```

```
function buildLightBoxOverlay( d , s , jqXHR ) { ... } ;

$('a.img-lk').click(function( event ) {
      event.preventDefault() ;

      var url = $(this).attr('href');
      var pr = $.ajax(url , {
                    type : "GET",
                    context: this,
                    xhrFields: { withCredentials : true }
                  });
      pr.done( buildLightBoxOverlay ) ;
      pr.fail( function(jqXHR, status, error) {
                    alert( "error loading data : "+ error) ;
                  } ) ;

});
```

des raccourcis pour les cas simples

- **$.get**(url, data) : requête GET sur l'url transmise, retourne une promise
- **$.getJSON**(url, data) : fait un GET pour récupérer des données JSON ; retourne une promise
- **$.post**(url, data) : requête POST sur l'url transmise, retourne une promise