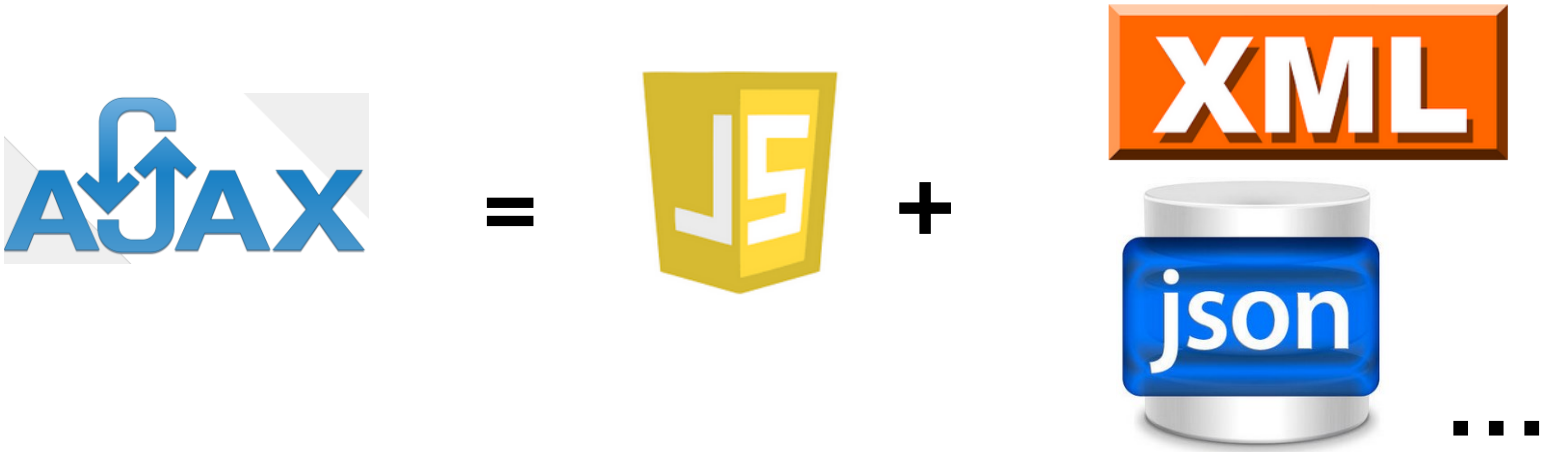
Développement Front-End (suite Partie 2)

AJAX

Aous Karoui
aous.karoui@iut2.univ-grenoble-alpes.fr

AJAX - Acronyme

Asynchronous Javascript And Xml



Création de compte

Gagnez du temps! Remplissez votre formulaire avec votre compte Facebook.

Mes identifiants :

Adresse e-mail* jloriaux@gmail.com

Confirmation d'adresse e-mail*

Mot de passe*

Confirmation de mot de passe*

Gagnez du temps ! Retrouvez vos informations en saisissant votre **numéro de client** et votre **nom de famille**.

N° de client

Nom OK

Mes coordonnées personnelles :

À chaque commande, vous pourrez choisir une autre adresse de livraison.

Titre* Madame Mademoiselle Monsieur

Veuillez indiquer votre civilité.

Nom*

Prénom*

N° et rue*

N° Appt, étage

Bâtiment

Inscription

?

AJAX en natif: Traiter la réponse

- On programme la fonction écouteur pour traiter la réponse, suivant le type de la réponse :

```
function fctRequeteOk(evt) {
  var contenuHTML= this.responseText;
  var contenuJson= JSON.parse(this.responseText);
ou var contenuJson= this.response;  //si responseType="json"
  var contenuXML= this.responseXML.documentElement;
ou var contenuXML= this.response;  //si responseType="document"
  Ensuite on traite la réponse
}
```

- Pour XML on peut utiliser l'API DOM

```
var arbreXML=xhr.responseXML.documentElement;
var unEtu=arbreXML.querySelector("etudiant");
var prenom=unEtu.querySelector("prenom").innerHTML;
var grp= unEtu.firstElementChild.getAttribute("groupe");
```

AJAX en jQuery: Traiter la réponse

- En spécifiant le “callback” dans les paramètres

```
$.ajax('script.php', {
        method: "POST",
        data: {
            param: value
        },
        success: traiterResultat
    });
```

- Librairie Javascript open-source
- parcours et modification du DOM
- événements
- effets visuels et animations
- manipulations des feuilles de style en cascade (ajout/suppression des classes, d'attributs…)
- Ajax

Rappel

1. Téléchargement

- http://jquery.com/download/

2. Installation

```
<script language="javascript" type="text/javascript" src="jquery-1.5.min.js"></script>
```

3. Application

- http://api.jquery.com/

Memento Jquery (sur Umtice)

Asynchrone

- Non synchrone avec le chargement initial de la page HTML
- On peut exécuter du code pendant le chargement
- On traite les données “quand elles arrivent”

Javascript

- Déclenche la “requête” vers le serveur
- Traite les données en retour

XML

- Utilisé comme format d’échange de données
- Dans le nom historiquement mais non obligatoire

AJAX - Fonctionnement

Schéma classique:

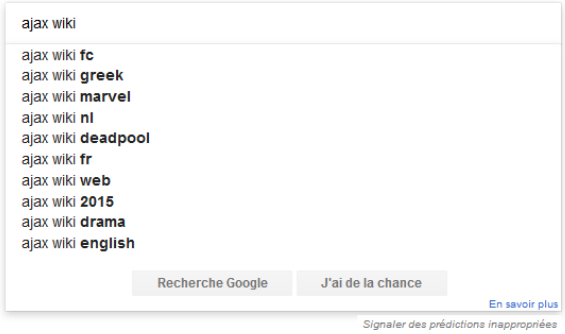
1. Requête HTTP initiale
2. Réponse du serveur (contenu complet)
3. Affichage direct de la réponse dans le navigateur

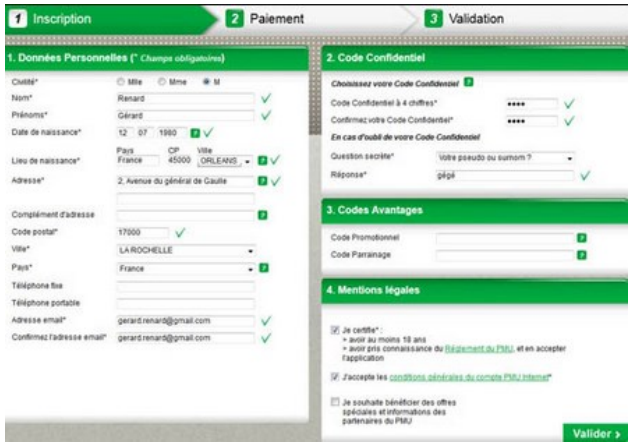
Version AJAX:

1. Requête HTTP initiale
2. Réponse du serveur (contenu complet)
3. Affichage direct de la réponse dans le navigateur
4. Requête AJAX (JS)
5. Réponse du serveur
6. Mise à jour de la page (JS)
7. Retour au 4

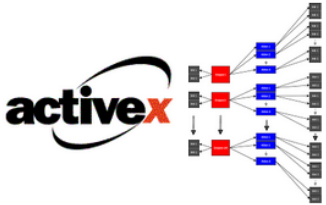
Exemples d'usages

- Auto-complétion
- Sauvegarde automatique
- Messagerie instantanée
- Formulaires avancés
- Chargement au fur et à mesure de contenu volumineux
- Upload de fichiers
- Etc.





AJAX - Histoire

- Initié par Microsoft sous le nom de 

- Généralisé par le W3C

- Deux versions XHR1 et XHR2 du même objet **XMLHttpRequest**
  - Suivant la version du navigateur, vous êtes en XHR1 ou XHR2
  - Actuellement, tous les navigateurs sont en XHR2 Affichage direct de la réponse dans le navigateur

- En fonction de la version d'IE, il faudra employer ActiveX XMLHttp au lieu du XMLHttpRequest
  - Avant IE7, on utilisait Microsoft.XMLHTTP
  - Après IE7, on utilise Msxml2.XMLHTTP

XHR - Écouteur

Les événements que l'on peut écouter sur cet objet

- **loadstart** : La requête démarre
- **abort** : Quand la requête est avortée (par exemple lors de l'appel de la méthode abort)
- **error** : La requête s'est mal terminée
- **load** : la requête s'est bien terminée
- **timeout** : la requête a dépassé le temps limite d'exécution
- **loadend** : la requête est terminée (avec succès ou non)
- **progress** : permet d'être averti de l'avancée de la requête (on peut calculer le taux d'avancement à l'aide des propriétés loaded et total de l'objet ProgressEvent reçu en paramètre)

XHR - Propriétés

- **responseText** : réponse reçue dans un format text (String)
- On peut se servir de cet attribut lorsque la réponse correspond à du code html, du code javascript ou bien encore au format JSON. Et évidemment pour du texte

- **responseXML** : réponse reçue dans un format XML
- Cette réponse est alors un DocumentObject Model et se manipule comme tel (voir API DOM)

- **responseType** : indique le type de réponse que l'on souhaite recevoir ("text" ou "","json","arraybuffer", "blob", "document")
- **response** : réponse reçue "convertie" en fonction du responseType. Si la conversion n'est pas possible, la propriété aura pour valeur null

- **status** : code http du résultat de la requête : 200 ok – 404 non trouvée, etc
- **statusText** : libellé du code HTTP

- **timeout** : permet de paramétrer le temps maximal d'exécution de la requête
- par défaut a pour valeur 0 (pas de timeout)

AJAX – Formats de réponse

du texte

une portion de code HTML

une image

un objet ou un tableau d'objets JSON

du code XML

du code javascript

des données BLOB : Binary large object

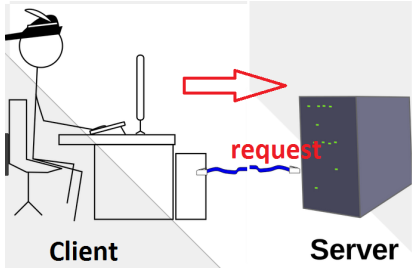
type de donnée permettant le stockage de données binaires
(le plus souvent des fichiers de type image, son ou video)

XHR - Méthodes

Les méthodes dont dispose cet objet

- **open** :permet d'ouvrir une requête AJAX
- **send** : permet de soumettre une requête
- **abort** : permet d'arrêter une requête en cours d'exécution
- **setRequestHeader** : permet de définir une entête HTTP

XHR – Envoyer une requete (Natif Ajax)



- Créer un objet XHR
  → var xhr = new XMLHttpRequest() ;
- Ouvrir la connexion avec le serveur
  → xhr.open(method, requête, asynchrone)
  avec **method**="GET " ou "POST", requête : chemin de la requête à exécuter sur le serveur , **asynchrone** : true/false

- Lorsque la requête est **asynchrone** : paramétrer les fonctions **écouteurs** à exécuter lorsque la requête se termine :
  xhr.addEventListener("load",fctRequeteOk) ;
- Si risque d'erreur :
  xhr.addEventListener("error",fctRequeteNOk) ;
- On soumet la requête :
  xhr.send(null) si méthod "GET"
  xhr.send(données) si méthod "POST"

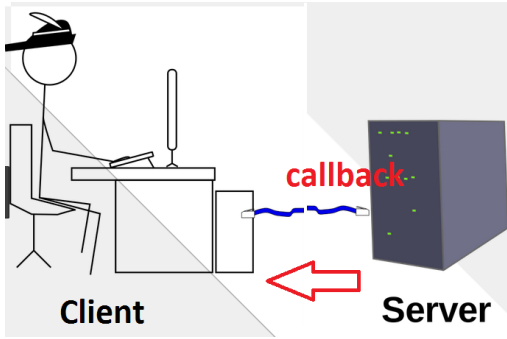
XHR – Traiter la réponse (Natif Ajax)

- On programme la fonction écouteur pour traiter la réponse, suivant le type de la réponse :

```
function fctRequeteOk(evt) {
  var contenuHTML= this.responseText;
  var contenuJson= JSON.parse(this.responseText);
  ou bien var contenuJson= this.response;  //si responseType="json"
  var contenuXML= this.responseXML.documentElement;
  ou bien var contenuXML= this.response;  //si responseType="document"
  Ensuite on traite la réponse
}
```

- Pour XML on peut utiliser l'API DOM

```
var arbreXML=xhr.responseXML.documentElement;
var unEtu=arbreXML.querySelector("etudiant");
var prenom=unEtu.querySelector("prenom").innerHTML;
var grp= unEtu.firstElementChild.getAttribute("groupe");
```



XHR – Envoyer des données (Natif Ajax)

- On utilise généralement un objet **FormData** pour transmettre les données en paramètre de la méthode send

```
var donnees=new FormData();

donnees.append("nom","smith");
donnees.append("prenom","Alfred");

xhr.send(donnees);
```

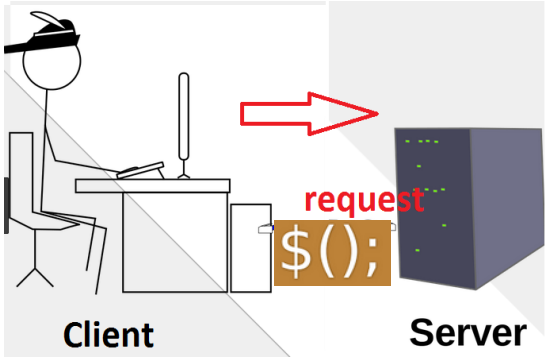
AJAX en jQuery: $.ajax

.ajax(), une instruction de bas niveau

Ne s’applique pas à un objet jQuery → `$('p').addClass('surbrillance');`

S’applique directement sur la librairie → `$.ajax({})`

AJAX en jQuery: $.ajax



- On utilise la méthode jQuery: **$.ajax**

```
$.ajax({
        url: "donnees-jquery.php",
        type:"POST",
        data: {
        param: value
        }
})
```

- $.ajax(url, parametres)
- Les paramètres sont précisés dans la documentation
- method indique le verbe HTTP (POST ou GET)
- data contient les paramètres de la requête HTTP
- Peu importe si GET ou POST

AJAX en jQuery: Traiter la réponse

```
$.ajax({
        url: "donnees-jquery.php",
        type:"POST",
        data: {
        param: value
        },
        success: traiterResultat
});
```

En spécifiant le "callback" dans les paramètres

```
$.ajax({
        url: "donnees-jquery.php",
        type:"POST",
        data: {
        param: value
        }

}).done(traiterResultat);
```

En utilisant le retour de Ajax avec done()

AJAX en jQuery: Traiter la réponse

Les paramètres:

- succes
- error
- complete

```
$("#more_com").click(function(){

    $.ajax({
        url : 'more_com.php',
        type : 'GET',
        dataType : 'html',
        success : function(code_html, statut){

        },

        error : function(resultat, statut, erreur){

        },

        complete : function(resultat, statut){

        }

    });

});

```

AJAX en jQuery: Traiter la réponse (autre exemple)

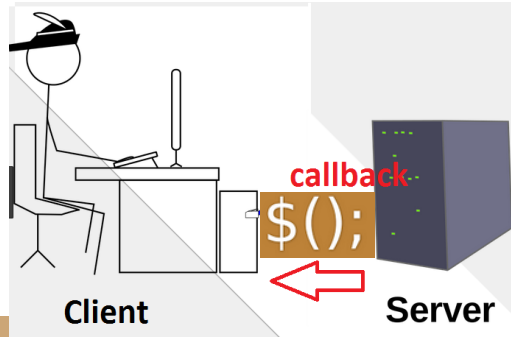
```
$(document).ready(function(e) {
    $.ajax({
                url: "ajax-jquery.php",
                complete:[retour1, retour2]
            }).done(function( arg ) {
                        alert( "Données : " + arg );
            });
});

function retour1(){
            alert("retour 1")
}
function retour2(){
            alert("retour 2")
}
```

AJAX en jQuery: Traiter la réponse

- Le type de donnée en retour est normalement automatiquement détecté
- Dans le cas XML on peut utiliser l'API jQuery

```
function traiterResultat(data) {
        var arbreXML= $(data);
        var unEtu=arbreXML.children("etudiant");
        var prenom=unEtu.children("prenom").html();
        var grp= unEtu.children().first().attr("groupe");
}
```



AJAX – Mise en pratique

1. Installer un serveur web local

2. Créer un script Php simple (affichage « echo; »)

3. S’adresser au fichier Php depuis notre fichier JS

AJAX – Mise en pratique

```
var obj = {envoi:'coucou'}
$.ajax({
        url: "donnees-jquery.php",
        type:"POST",
        data:obj
}).done(function( arg ) {
        alert( "Données : " + arg );
});
```

```
<?php
        $retour = $_POST['envoi'];
        echo $retour;
?>
```