Formation interne AJAX

TABLE DES MATIERES

Table des matières

I - AJAX - Généralités

A - Définitions

Ajax est une technique qui fait usage des éléments suivants:

- (X)HTML : pour la mise en page
- CSS pour la présentation de la page.
- Javascript pour les traitements locaux, et DOM (Document Object Model) qui accède aux éléments de la page ou du formulaire ou aux éléments d'un fichier xml pris sur le serveur (avec la méthode getElementsByTagName par exemple)... L'objet XMLHttpRequest lit des données ou fichiers sur le serveur de façon asynchrone.
- Si besoin, DOMparser intègre un document XML.

PHP ou un autre langage de scripts peut être utilisé coté serveur.

Le terme "Asynchronous", asynchrone en français, signifie que l'exécution de Javascript continue sans attendre la réponse du serveur qui sera traitée quand elle arrivera. En mode synchrone, le navigateur serait gelé en attendant la réponse du serveur.

A titre de bonus, voyez un effet CSS associé à un effet de scriptaculous en plaçant votre curseur au dessus du titre général "Formation interne à Ajax"...

B - Comment çà marche ?

1) Contrairement au fonctionnement habituel

- on clique sur un lien,
- on envoie une requête au serveur
- celui-ci envoie une réponse au navigateur
- le navigateur affiche la page (entière)
- on est déconnecté du serveur

2) en Ajax :

- Ajax utilise un modèle de programmation comprenant d'une part la présentation, d'autre part les évènements.
  Les évènements sont les actions de l'utilisateur, qui provoquent l'appel des fonctions

associées aux éléments de la page.
L'interaction avec l'utilisateur se fait à partir des formulaires ou boutons html.

- Ces fonctions JavaScript identifient les éléments de la page grâce au DOM et communiquent avec le serveur par l'objet XMLHttpRequest.
  Pour recueillir des informations sur le serveur cet objet dispose de deux méthodes:
  - open: établit une connexion.
  - send: envoie une requête au serveur.
- Les données fournies par le serveur seront récupérées dans les champs **responseXml** ou **responseText** de l'objet XMLHttpRequest. S'il s'agit d'un fichier xml, il sera lisible dans responseXml par les méthodes de Dom.
- Noter qu'il faut créer un nouvel objet XMLHttpRequest, pour chaque fichier que vous voulez charger.
- Il faut attendre la disponibilité des données, et l'état est donné par l'attribut **readyState** de XMLHttpRequest.
  Les états de **readyState** sont les suivants (seul le dernier est vraiment utile):
  - 0: non initialisé.
  - 1: connexion établie.
  - 2: requête reçue.
  - 3: réponse en cours.
  - 4: terminé.

C - En pratique

Il faut se souvenir que les requêtes Ajax ne sont pas les seules à être envoyées par le navigateur vers un ou plusieurs serveurs, de façon transparente pour l'utilisateur et sans rechargement de la page :

C'est le cas des images, des feuilles de styles, des scripts externes... pour lesquels le navigateur va soumettre une requête au serveur concerné.

Toutefois, ces requêtes sont envoyées sans intervention de l'utilisateur et elles sont en nombre limité (par défaut : 10 simultanées dans I.E.)

Ceci explique parfois des temps de chargement longs pour une page si elle fait appel à de trop nombreux fichiers externes, en particulier à des images de grande taille...

Pour revenir à notre sujet, on utilisera surtout de l'"AJAT" c'est à dire que la réponse du serveur sera envoyée sous forme texte et non sous la forme d'un document XML.

Ce texte contiendra le plus souvent du code XHTML avec ses balises de mise en forme, voire ses styles associés.

Pour mettre à jour la page, il suffit d'injecter la réponse de type texte dans la balise de destination (propriété innerHTML de la balise).

C'est d'ailleurs la méthode utilisée par les "frameworks" prototype et jQuery (pour ne citer qu'eux).

Avec une réponse au format XML, le navigateur doit encore appliquer du code javascript qui devra

traiter l'arborescence du document XML reçu, avec une syntaxe qui, même si elle n'est pas insurmontable, reste délicate à manipuler...

D - Ce qui vous est proposé :

- La première partie de cette formation traite des mécanisme de base, du fonctionnement d'AJAX.
- La deuxième partie présente l'utilisation de bibliothèques ("frameworks") qui simplifient la programmation en encapsulant le travail de bas niveau et en offrant des possibilités très intéressantes pour le code Javascript. La première bibliothèque présentée est **prototype.js**.
- La troisième partie montre des effets générés par la bibliothèque **scriptaculous.js** associée à prototype.
- La quatrième partie montre des effets générés par la bibliothèque **Jquery** Cette bibliothèque est plus récente, elle est en évolution (contrairement à prototype, dont le développement semble avoir été arrêté depuis début 2008). Associée à **jQuery_ui**, elle propose des effets visuels et des plug-in intéressants (très nombreux, disponibles sur internet).
- Elément important pour notre équipe : si le CMS choisi pour les sites intranets est bien DRUPAL, celui-ci s'appuie sur le framework jQuery. C'est donc une incitation supplémentaire pour s'y intéresser de plus près.

II - L'objet XHR

Cette présentation est largement inspirée du site "Les standards du Web", elle a été simplifiée en retirant des détails de moindre importance pour notre formation :

A - Historique

XMLHttpRequest a en premier lieu été développé par Microsoft, comme un objet ActiveX, pour Internet Explorer 5. Il a ensuite été repris et implémenté successivement sous Mozilla, Safari , Konqueror et dernièrement Opera .

À cause de ces implémentations plutôt récentes, l'objet n'est pas supporté par les navigateurs dits de «vieille génération».

En avril 2006, il a été proposé pour devenir une **recommandation du W3C**.

L'utilisation de XMLHttpRequest nécessite l'utilisation du langage **javascript**.

Les navigateurs l'implémentent de différentes façons : ActiveX pour Internet Explorer, objet pour les autres. La création de l'objet se fait par conséquent différemment selon le navigateur. Qui plus est , pour créer l'ActiveX sous Internet Explorer, il est nécessaire de tester plusieurs versions.

Cependant, après la création de l'objet, les méthodes et attributs sont les mêmes pour l'ensemble des navigateurs. Donc comme d'habitude, I.E. se singularise par une syntaxe plus complexe : il a incorporé des fonctionnalités XMLHttp dans Internet Explorer sous forme d'un ActiveX (IE < 7) sous le nom MSXML.

L'usage des "frameworks" permet de s'affranchir de ces différences en encapsulant les différentes étapes de cette création.

Les développeurs du projet mozilla ont alors implémenté leur version de XMLHTTP sous le nom XMLHttpRequest.

Le World Wide Web Consortium essaie de standariser les API en publiant les spécifications 1 et 2 sous le nom XMLHttpRequest.

Ainsi plusieurs implémentations de XMLHttpRequest existent. Pour supporter toutes ces versions une API javascript (AJAX) de haut niveau sert à faire abstraction des différentes implémentations et versions de XMLHttpRequest .

La communauté des développeurs a coutume d'abréger le terme en **XHR**

B - Avantages et inconvénients

1) L'avantage principal

est dans le mode de fonctionnement asynchrone. La page entière ne doit plus être rechargée en totalité quand une partie doit changer ce qui entraîne un gain de temps et une meilleure interaction avec le serveur et par conséquent le client.

Par ailleurs, la charge de travail est plus équitablement répartie entre le navigateur et le serveur, surtout depuis l'avènement de moteurs javascript rapides dans les dernières versions des navigateurs (I.E. est encore largement à la traîne, surtout en comparaison avec Chrome, Opéra et même Firefox).

2) Les inconvénients

résident dans le fait que

- XMLHttpRequest en version actuelle ne peut se connecter qu'à un seul domaine serveur en utilisant le protocole http
- que le navigateur doit être compatible.

Le contenu dynamique affiché par l'utilisation de javascript explique pourquoi le navigateur ne peut pas avoir le même comportement qu'une page html chargée dans sa totalité (enregistrement des liens, marque page, bouton retour).

D'autres différences existent :

- le débogage du javascript est difficile à moins d'intégrer dans Firefox des outils comme Firebug.
- le nombre de requêtes pouvant s'exécuter en même temps dépend du navigateur.

C - création de l'objet XHR

```
1. function getXhr(){
2.   var xhr = null;
3.   if(window.XMLHttpRequest) {// Firefox et autres
4.     xhr = new XMLHttpRequest();
5.   } else if(window.ActiveXObject){ // Internet Explorer
6.     try { // deux activeX possibles selon les versions
7.       xhr = new ActiveXObject("Msxml2.XMLHTTP");
8.     } catch (e) {
9.       xhr = new ActiveXObject("Microsoft.XMLHTTP");
10.         }
11.       }
12.       else { // XMLHttpRequest non supporté par le navigateur
13.         alert("Votre navigateur ne supporte pas les objets
   XMLHTTPRequest...");
14.         xhr = false;
15.       }
16.       return xhr;
17.     }
```

en ligne 3 : on teste si l'objet standard existe (Firefox et autres navigateurs respectant les normes)

si c'est le cas (ligne 4) on crée cet objet xhr que l'on retournera plus tard.

ligne 5 : sinon, si on est dans I.E. avec un activeXObject, il reste encore à tester duquel il s'agit d'où les lignes 6 à 10.

Si aucun objet n'a été trouvé, le navigateur est trop ancien, il ne reste qu'à prévenir l'utilisateur qu'il ne pourra pas bénéficier des fonctionnalités d'AJAX.

La fonction getXhr() renvoie un objet requête que l'on pourra manipuler par la suite.

Le code de cette fonction étant réutilisable, il est judicieux de le déporter dans un fichier js externe...

III - TUTORIEL AJAX - La requête

A - Création de la requête

Une fois l'objet xhr obtenu par l'appel à la fonction précédente (getXHR(); on peut manipuler ses propriétés et ses méthodes :

Les propriétés et méthodes seront naturellement préfixées par le nom de l'objet requête (par exemple : myXhr = getXHR();

myXhr.readyState …

Les propriétés et méthodes principales sont en ***gras-italique***

Propriété ou méthode	Description des paramètres
open("méthode","url",flag)	Ouvre la connexion avec le serveur. **méthode** -> "GET" ou "POST" **url** -> l'url à laquelle on va envoyer notre requête. Si la méthode est GET, on met les paramètres dans l'url, sous la forme *url?nom1=valeur 1&nom2=valeur 2* **flag** -> true si l'on veut un dialogue asynchrone, sinon, false
setRequestHeader("nom","valeur")	Assigne une valeur à un header HTTP qui sera envoyé lors de la requête. Obligatoire pour utiliser la méthode **POST** : nom -> "Content-Type" valeur -> "application/x-www-form-urlencoded"
send("params") => POST ***send***(null) => GET	Envoie la requête au serveur. Si la méthode est **GET**, on met **null** en paramètre. Si la méthode est **POST**, on met les paramètres a envoyer, sous la forme : *"nomparam1=valeurparam1&nomparam2=valeurparam2"*.
abort()	Abandonne la requête.
onreadystatechange	Ici, on va désigner la fonction qui sera exécutée à chaque "changement d'état" de notre objet. Notez bien la casse : il n'y a pas de majuscules

readyState	C'est cette propriété qu'on va tester dans le onreadystatechange. Elle représente l'état de l'objet et peut prendre plusieurs valeurs : 0 -> Non initialisé. 1 -> Ouverture (open() vient de s'exécuter). 2 -> Envoyé (send() vient de s'exécuter). 3 -> En cours (des données sont en train d'arriver). 4 -> Prêt (toutes les données sont chargées) on peut exploiter le résultat. Attention au S majuscule
status	Le code de la réponse du serveur. 200 -> OK. 404 -> Page non trouvée. ...
statusText	Le message associé à status.
responseText	La réponse retournée par le serveur, au format texte.
responseXML	La réponse retournée par le serveur, au format DOM XML.

Exemple

```
1. function go(){
2.   var xhr = getXhr();
3.   xhr.onreadystatechange = function(){
4.     if(xhr.readyState == 4 && xhr.status == 200){
5.       alert(xhr.responseText);
6.     }
7.   }
8.   xhr.open("GET","ajax.php",true);
9.   xhr.send(null);}
```

```
Ligne 2 :   On crée un objet xhr.
Ligne 3 :   On définit la fonction à exécuter :
Ligne 4 :   Si l'état est "prêt" et si la réponse est OK
Ligne 5 :   alors, on peut exécuter le traitement...
Ligne 8 :   on établit la connexion (open)
ligne 9 :   on envoie la requête (send)
```

B - exemple de requête simple

On dispose d'un fichier texte (lorem.txt) dont on veut faire afficher le contenu dans une balise de notre page :

Ce fichier texte est situé dans un sous-dossier "serveurs"

La balise de destination (qui peut contenir un texte d'attente) s'appelle "here"

Voici le code javascript qui convient :

```
1. function lorem(){
2.   var xhrLorem = getXhr();
3.   xhrLorem.onreadystatechange = function(){
4.     if(xhrLorem.readyState == 4 && xhrLorem.status == 200){
5.       document.getElementById("here").innerHTML =
   xhrLorem.responseText;
6.     }
7.   }
8.   xhrLorem.open("GET","serveurs/lorem.txt",true);
9.   xhrLorem.send(null);}
```

Ce code sera exécuté lorsque l'événement onclick sera intercepté sur un bouton...

<button onclick="lorem();">....

C - mode synchrone et asynchrone

Par défaut, les requêtes AJAX sont prévues pour fonctionner en mode asynchrone : pendant que le serveur exécute la requête et jusqu'à ce que la réponse soit complète, le navigateur continue son interprétation de la page et le javascript est actif.

La page n'ayant pas à se recharger, l'aspect dynamique se fait sans blocage de l'utilisateur.

Dans certains cas cependant, ceci peut présenter des inconvénients : par exemple si la réponse crée des éléments sur la page, ceux-ci ne seront pas accessible tant que la requête précédente ne sera pas terminée et traitée. Dans ces circonstances, il faudra faire appel à une requête synchrone : le navigateur reste en suspens jusqu'à la fin du traitement de la requête courante.

Il existe assez souvent des moyens de contourner cette difficulté, comme nous le verrons plus loin : une requête doit ramener les noms des villes commençant par les caractères saisis par l'utilisateur.

La table des villes comporte près de 35000 entrées, il est évident que le temps de traitement risque d'être long ! Dans ce cas, on peut limiter le nombre de réponses à fournir par le serveur (nous avons choisi de ne ramener que les 10 premiers noms correspondant au critère de sélection). Le temps de traitement est alors notablement plus court et redevient compatible avec une requête asynchrone.

IV - Ajax avec prototype

A - Généralités

Nous avons vu qu'Ajax s'appuie sur le javascript pour la partie fonctionnant sur le navigateur.

Par ailleurs, la création de l'objet XHR est une opération initiale dont la complexité mérite d'être masquée (merci I.E. !!!).

Le javascript utilise également de plus en plus la syntaxe héritée du DOM et la répétition d'instructions du genre "document.getElementById("xyz")" devient fastidieuse...

Divers auteurs ont donc créé des bibliothèques (ou frameworks) pour encapsuler ces difficultés et arriver à un langage finalement plus simple (une fois les fonctionnalités de la bibliothèque assimilées).

C'est le principe du "write less, do more" : faites-en plus en écrivant moins.

Il est évident que `$F("prenom")` est plus pratique que `document.getElementById("prenom").value` !

et $("zone2").show() est plus lisible que
document.getElementById("zone2").style.display="block";

Pour AJAX, on va encore plus loin car les bibliothèque proposent non seulement des fonctions "de bas niveau" donnant un contrôle complet sur les XHR mais fournissent aussi des mécanismes "tout prêts" pour les usages les plus fréquents.

Nous avons retenu deux bibliothèques dans le cadre de cette formation :

* prototype (et son compagnon scriptaculous)

* jQuery (et son acolyte jQuery_ui).

Il en existe bien d'autres (Dojo, Mootools etc) mais elles répondent toutes plus ou moins aux mêmes principes et rendent à peu près les mêmes services.

Prototype (et scriptaculous) sont toujours très utilisées sur les sites Web, cependant, il semble que leur développeur ait cessé de maintenir son produit depuis le début de 2008.

jQuery (et jQuery_ui) sont toujours en évolution et sont associées à des plug-in de plus en plus nombreux disponibles sur internet, gratuitement pour la plupart.

B - prototype.js

La librairie prototype.js est un ensemble de fonctions avancées qui améliorent l'écriture du code Javascript en le simplifiant :

- grâce à des utilitaires de manipulation du DOM
  - **$("id")** => document.getElementById("id")
  - **$$(selecteur)** => tableau d'éléments répondant à la description du sélecteur CSS par exemple : $$(div#menu ul ul li a) renvoie un tableau des liens contenus dans les items de listes de second niveau appartenant à la balise div id="menu" !
  - **$F("element"**) remplace avantageusement document.getElementById("element").value pour un élément de formulaire !
  - **document.getElementsByClassName(nomAttribut[,racine])** renvoie un tableau d'éléments possédant un attribut donné,
    racine (facultatif) indique l'élément racine de la recherche.
- grâce à des objets qui encapsulent l'utilisation d'Ajax :
  - **Ajax.Updater** est certainement l'objet le plus utile car il fonctionne aussi bien sous I.E. que sous les autres navigateurs.
    Cet objet prend en charge toutes les étapes relatives à la requête xhr, jusqu'à l'alimentation de la balise cible.
    Par contre, cet objet ne sait traiter que des réponses de type texte.
    - Exemple :

```
new Ajax.Updater('resultat', '../serveurs/identite.php', {
  parameters: { prenom: $F('prenom') },
  insertion: Insertion.Bottom
});
```

    - ce code va interroger le script "identite.php" situé dans un dossier "serveurs" en lui transmettant (par la méthode POST) le couple variable=valeur, variable étant ici "prénom" et la valeur est lue dans l'élément de formulaire nommé "prenom".
    - Le résultat renvoyé (en responseText) par le serveur sera ajouté à la fin de la balise dont l'id est "resultat".
  - **Ajax.PeriodicalUpdater** permet de lancer à intervalles réguliers un Ajax.Updater. Le délai entre deux appels peut être allongé tant que les réponses sont identiques pour éviter de surcharger le réseau avec des requêtes inutiles.
- De nombreux autres modules existent, en particulier pour les chaines de caractères, mais ils sont en dehors du cadre de cette formation.

1) exemples de codes avec prototype

Pour charger la bibliothèque, il suffit d'inclure dans la section head la balise suivante
<script type="text/javascript" src="../js/prototype.js"></script>
ATTENTION : Il ne faut pas utiliser la méthode de fermeture interne de la balise script, sinon le code ne fonctionnera pas.

Une fois la bibliothèque chargée, vous disposez de ses facilités d'écriture dans votre javascript!

a) exemple 1 : changer les caractéristiques d'un paragraphe :

Le paragraphe dans la page : en cliquant dans son texte, celui-ci passera en caractères gras

```
<body>
<p onclick="gras(this);">Cliquez-moi pour mettre mes caractères en gras </p>
</body>
```

La fonction javascript dans le <head>

```
function gras(element) {
  // met en gras le contenu de 'element'
    $(element).style.fontWeight="900";
  }
```

b) utilisation des sélecteurs CSS pour identifier un groupe d'objets :

Le sélecteur * p+p ramène pour tous les éléments (*) les paragraphes descendants qui sont en même temps précédés par un paragraphe.

```
function pSuivant() {
  // cette fonction change la graisse du texte de tous les paragraphes précédés d'un autre paragraphe
  pp=$$("* p+p"); // retourne un tableau des paragraphes qui suivent un autre paragraphe
  for (i=0;i<pp.length;i++) {
    pp[i].style.fontWeight="900";
  }
}
```

pp contient un tableau formé par tous les paragraphes qui suivent un autre paragraphe. Pour chacun d'entre eux (boucle for), on modifie dans les caractéristiques de leur style, la valeur de la propriété fontWeight.

ATTENTION : en CSS, on écrit font-weight. En javascript, on traduit cela en fontWeight (notation camélisée).

c) exemple avec Ajax.Updater

Le code javascript est simplifié au maximum, pas besoin de créer l'objet xhr, de définir la fonction de traitement...
ATTENTION : la méthode POST est utilisée pour l'appel de la requête (en tenir compte dans le code du serveur)

Une liste déroulante (<select>) est initialisée par une requête PHP, elle contient la liste des régions (numéro en value, nom en étiquette).

On intercepte l'événement onchange pour appeler une fonction javascript.
Cette fonction utilise AJAX pour demander au serveur de lui renvoyer, sous la forme d'une table HTML, la liste des départements de la région choisie.
Cette liste s'affiche dans une balise

Voici le code javascript de cette fonction :

```
<script type="text/javascript">
 function go() {
       new Ajax.Updater('ici', 'serveurs/regdep.php', {
       parameters: { reg: $F('selreg') },
       });
}
</script>
```

On peut constater la simplicité du code nécessaire !

V - Ajax avec prototype et scriptaculous

A - Généralités

La librairie scriptaculous.js est un ensemble de fonctions qui apportent à Javascript des fonctions de manipulation graphique des éléments de la page:

On trouve des effets de base (Morph, Move, Opacity, Parallel, Scale, Highlight) et des combinaisons d'effets (Appear, BlindDown, BlindUp, DropOut, Fade, Fold... ainsi qu'un effet de bascule : toggle)

Scriptaculous propose également des méthodes pour gérer le Drag&Drop, l'autocomplétion et l'édition sur place (InPlace Editing)

B - Chargement des scripts

Scriptaculous utilise prototype, cette bibliothèque doit donc être chargée en premier lieu :

```
<script type="text/javascript" src="../js/prototype.js"></script>
<script type="text/javascript" src="../js/scriptaculous.js"></script>
```

Pour le détail de ces effets, reportez-vous au fichier *.chm téléchargé sur le site ou sur les nombreux tutoriels disponibles sur le Web...

VI - Ajax avec jQuery

En préambule :

le framework jQuery est utilisé par le gestionnaire DRUPAL.

Si son choix se confirme pour la gestion des sites intranets de la DGFiP, un minimum de connaissances de jQuery (et de jQuery_ui) seront indispensables à acquérir !

A - La méthode de base : $.ajax()

$.ajax() retourne l'objet XMLHttpRequest qu'il crée. Dans la plupart des cas, vous n'aurez pas besoin de manipuler cet objet directement, mais vous pouvez l'utilisez si vous souhaitez annuler une requête manuellement;

Pour un niveau d'abstraction plus grand et plus facile à utiliser, consultez les fonctions $.get ou $.post etc... A noter cependant que ces dernières fonctions ne proposent pas autant de fonctionnalités que $.ajax()

1) syntaxe et éléments de la fonction :

a) La syntaxe de départ :

$.ajax(); // renvoie un objet XHR mais ne fait rien d'autre.

$.ajax({*options*}); // les options indiqueront l'URL du serveur, les paramètres à lui envoyer etc... Elles sont présentées un peu plus loin.

b) Un exemple simple :

```
var html = $.ajax({
 url: "some.php"
}).responseText;
```

$.ajax({url: "some.php"}) renvoie un objet XHR qui a interrogé le serveur "some.php" avec une méthode POST et en asynchrone par défaut.

Le type de réponse est déterminé par jQuery selon la nature des données reçues (texte, XML, JSON …)

Une fois cet objet créé et la requête terminée, on peut alors récupérer la réponse sous forme de

texte et l'attribuer à une variable "html".

c) les options :

- **async** (booléen): par défault, toutes les requêtes sont asynchrones (ce paramètres vaut true par défaut). Si vous avez besoin de requêtes synchrones, passez cette fonction à false.
  A noter que les requêtes synchrones bloquent temporairement le navigateur de l'utilisateur tant que la requête n'est pas terminée.
- **beforeSend** (fonction): un pre-callback permettant de modifier l'objet XMLHttpRequest avant qu'il soit envoyé. A utiliser pour envoyer des entêtes personnalisés par exemple etc. Seul l'objet XMLHttpRequest est passé en argument de cette fonction.
  Cf en d) l'exemple détaillé avec l'emploi d'une image d'attente.
- **complete** (fonction): fonction à appeler lorsque la requête se termine (après que les callbacks de succès et d'erreurs soient exécutés).
  La fonction dispose de deux arguments:
  l'objet XMLHttpRequest
  et une chaine de caractères décrivant le type de succès de la requête.
- **contentType** (String): Quand vous envoyez des données au serveur, utilisez ce paramètre. Par défaut, il vaut "application/x-www-form-urlencoded", ce qui correspond dans la plupart des cas.
- **data** (objet|string): Donnée à envoyer au serveur. Elle est convertie en String, si elle ne l'est pas déjà. Consultez l'option processData pour empêcher ce processus automatique.
  L'objet doit être formé de paires de la forme clé/valeur.
  Si la valeur est un tableau, jQuery sérialise les différentes données du tableau avec la même clé. ***Par exemple: {foo:["bar1", "bar2"]} devient '&foo=bar1&foo=bar2'.***
  Ce paramètre n'est pas nécessaire pour les requêtes utilisant la méthode GET.
- **dataType** (string): format des données qui seront renvoyées du serveur. Si aucun type n'est spécifié, jQuery utilisera le type MIME pour déterminer le format adéquat: responseXMl ou responseText. Voici la liste des types disponibles:
  - "xml": retourne un document XML qui pourra être traité par jQuery.
  - "html": retourne du code HTML au format texte, inclus l'évaluation des script tags.
  - "script": évalue la réponse en Javascript et retourne cette dernière au format texte.
  - "json": évalue la réponse en JSON et retourne un objet Javascript.
- **error** (fonction): Fonction à appeler si la requête échoue.
  La fonction dispose de trois arguments:
  - l'objet XMLHttpRequest,
  - une chaine de caractère décrivant le type d'erreur rencontré,
  - et un objet d'exception, dans la cas ou ce dernier a été généré.
- **global** (booléen): permet le déclenchement du gestionnaire d'évènement global de AJAX. Par défaut, il vaut ***true***. Passez false à cette option si vous voulez empêcher les déclenchements d'événements de type ajaxStart ou ajaxStop.

- **ifModified** (booléen): la requête se termine avec succès seulement si les données retournées sont différentes de la dernière requête. Les entêtes sont utilisés pour cette opération. Par défaut, cette option vaut "false".
- **processData** (booléen): permet de ne pas passer en chaine de caractères les données passées à l'option "data". Si vous souhaitez envoyer des documents DOM, ou d'autres données non traitables, passez cette option à false.
- **success** (fonction): Fonction à appeler si la requête s'exécute avec succès. Un seul argument est passé en paramètre: les données retournées par le serveur, format suivant le format défini par l'option "dataType".
- **timeout** (entier): spécifie un timeout local en millisecondes pour la requête. Ce timeout prendra le pas sur le timeout global (défini par la fonction $.ajaxTimeout()) pour la requête.
- **type** (string): type de la requête (GET ou POST), par défaut, vaut GET. ***D'autres méthodes d'envoi HTTP peuvent être utilisées, comme PUT ou DELETE, mais celles-ci ne sont pas supportées par tous les navigateurs.***
- **url** (string): URL de la requête.

B - La méthode GET

La page sera chargée en utilisant la méthode GET, sans avoir besoin d'utiliser la fonction plus complexe $.ajax.

Cette fonction permet d'associer une méthode qui sera exécutée lorsque la requête sera effectuée. Si vous avez besoin de définir des callback d'erreurs et de succès, utilisez la fonction $.ajax.

1) syntaxe élémentaire

```
  $.get(URL,
    { nom: valeur },
    function(data){
      $("destination").html(data);
    }
  );
}
```

2) paramètres

nom et type	rôle
url (String)	URl de la page à charger
params (Map): (optionnel)	paires de clé/valeur qui seront envoyées au serveur

nom et type	rôle
callback (Fonction): (optionnel)	fonction qui sera exécutée quand les données seront chargées
type (String): (optionnel)	format des données renvoyées à la fonction de callback

3) exemple de code

```
function ajaxget(reg) {
  $.get("serveurs/get_regdep.php",
    { reg: reg },
    function(data){
      $("#resultat").html(data);
    }
  );
}
```

4) exemple d'utilisation

Une liste déroulante (balise select) est initialisée par une requête PHP à la création de la page.

Pour chacune de ses options, la valeur est l'identifiant de la région dans la table, l'étiquette est le nom de la région en clair.

On intercepte l'événement onchange sur lequel on appelle la fonction javascript précédente :

```
<select onchange="ajaxget(this.value)" name="selreg">
```

Dans le body, il suffit de placer une balise (div ou span) d'id="resultat". Cette balise recevra comme contenu la réponse donnée par le serveur.

C - La méthode POST

Le principe est identique mais la méthode utilisée est POST, ce qui permet d'envoyer plus de données au serveur.

Le code javascript est très peu modifié :

```
function ajaxPost(reg) {
  $.post("serveurs/post_regdep.php",
    { reg: reg },
    function(data){
      $("#resultat").html(data);
    }
  );
}
```

Le déclenchement est fait de manière identique sur l'événement onchange de la liste déroulante :

```
<select name="selreg" onchange="ajaxPost(this.value)">
```

VII - jQuery et jQuery_ui

Comme scriptaculous est le complément de prototype, jquery_ui est le complément graphique de jQuery.

Sa dernière version (1,8,3 à ce jour) a intégré les plugins les plus populaires, ce qui évite d'avoir à les ajouter dans la liste des fichiers js.

Par ailleurs, il existe un outil en ligne qui compile à la fois les effets et compléments désirés et également les feuilles de style qui construisent le thème de votre choix.

Dans le cadre de cette formation déjà largement fournie, nous ne nous étendrons pas sur ces possibilités supplémentaires mais je ne peux que vous inciter à vous y pencher à titre personnel...