Ajax - une autocomplétion pas à pas

par Denis Cabasson

Date de publication : 10/02/2006

Dernière mise à jour :

Ajax est un terme très à la mode actuellement dans les développements web. Cet article propose de créer pas à pas un script d'auto-complétion utilisant cette technologie, dans deux buts : créer une interface conviviale d'aide à la saisie (type Google Suggest) mais aussi et surtout voir ce qui se cache vraiment sous la dénomination d'Ajax.

Introduction

Ajax est un acronyme très à la mode de nos jours. Qui n'a pas rêvé de pouvoir rajouter le prestigieux label Ajax (Asyncronimous Javascript And Xml) à un de ses sites web? Le but de cet article est de reprendre pas à pas le mécanisme d'auto-complétion utilisant la technologie Ajax mis en place à de nombreux endroits. L'exemple le plus parlant, et peut-être le plus abouti est celui de Google Suggest qui vous propose au fur et à mesure que vous entrez le texte de votre recherche les complétions les plus populaires.

Pour ce genre de choses, il est bien entendu impensable de passer dès le chargement de la page web l'ensemble des informations aux navigateurs clients. Le volume que cela représenterait saturerait le serveur en un temps record! C'est là qu'intervient notre fameuse fée magique Ajax, qui va aller chercher l'information sur le serveur et l'intégrer au client sans que celui-ci ait à subir un rechargement de complet de la page.

Pour cet exemple, nous utiliserons un script très basique pour le serveur de données (une simple page PHP) car ce qui nous intéresse le plus est bien évidemment la mise en place de tout le moteur JavaScript nécessaire pour faire fonctionner notre auto-complétion. De plus, la compatibilité de Google Suggest avec les navigateurs anciens est très poussée. Dans le but de rendre les scripts plus lisibles, nous nous contenterons d'une version qui fonctionne avec les navigateurs récents (IE 6, Firefox 1.5 et partiellement Opera 8.5).

Nous allons créer un script permettant de gérer une liste déroulante, dans le sens habituel du terme, qui présentera les fonctionnalités qu'un utilisateur est en droit d'attendre d'une telle liste : proposition de complétion du champ texte, surlignement de la suggestion active, navigation par les flèches haut/bas ou à la souris, ...

Dans un souci de clarté, nous mettrons en oeuvre les fonctionnalités de notre script d'auto-complétion progressivement, en trois grandes étapes: La mise en place du dialogue client/serveur, la mise en place de la présentation et enfin la gestion des évènements.

1 - Etape 1 - Faire communiquer le client et le serveur

Cette première étape est celle ou nous allons mettre en oeuvre le fameux objet xmlHttpRequest qui va nous permettre de faire communiquer notre navigateur web avec le serveur, sans que l'utilisateur ait besoin de recharger sa page.

C'est le coeur de toute page utilisant Ajax, et la nouveauté introduite par cette technologie. Mais pour autant, ce n'est absolument pas la partie la plus difficile de la chose....

L'objet xmlHttpRequest permet comme son nom l'indique d'effectuer une requête HTTP vers notre serveur (et uniquement celui-là pour des raisons de sécurité), et d'effectuer un traitement dans notre page au moment du retour de la requête. Dans notre cas, la requête nous donnera les 10 premières possibilités de complétion de notre champ texte.

1-A - Coté serveur

Tout d'abord intéressons-nous à la page coté serveur qui va renvoyer les possibilités de complétion à notre utilisateur. Comme il ne s'agit pas vraiment du sujet de cet article, nous allons la réduire à son strict minimum de façon à ce qu'elle nous retourne, pour une entrée utilisateur donnée, un fichier XML contenant les complétions possibles.

Voici le fichier php que nous utiliserons:

```
<?php
header('Content-Type: text/xml;charset=utf-8');
echo(utf8_encode("<?xml version='1.0' encoding='UTF-8' ?><options>"));
if (isset($_GET['debut'])) {
    $debut = utf8_decode($_GET['debut']);
} else {
    $debut = "";
}
$debut = strtolower($debut);
$liste = array([...]);

function generateOptions($debut,$liste) {
    $MAX_RETURN = 10;
    $i = 0;
    foreach ($liste as $element) {
        if ($i<$MAX_RETURN && substr($element, 0, strlen($debut))==$debut) {
            echo(utf8_encode("<option>".$element."</option>"));
            $i++;
        }
    }
}

generateOptions($debut,$liste);

echo("</options>");
?>
```

La liste de mots utilisée (dans la variable $liste, qui n'a pas été recopiée ici) est la liste des 1500 mots maîtrisés par les élèves de CE2.

Vous pouvez tester le retour de cette page php: Liste des mots commençant par "de".

Le nombre maximum de possibilité retournée par cette page est 10. Dans une application réelle, on peut tout à fait envisager que les complétions les plus probables soient retournée en premier (comme pour Google Suggest). Ces traitements sont à la charge du serveur et à votre initiative.

Le seul point particulier de cette page php est de bien noter que, de façon générale, il convient d'envoyer la réponse XML en UTF-8 bien propre, pour éviter d'éventuels problèmes d'encodage par après (accents qui disparaissent, document XML non reconnu...).

1-B - Coté client

1-B-1 - La page HTML

Coté client, nous allons commencer par mettre en place une page HTML, la plus simple possible. Elle sera complétée par deux balises script qui respectivement la liera à notre script javascript et initialisera ce script.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
    <head>
        <title>Test d'autocompletion</title>
    </head>
    <body>
        <form name="form-test" id="form-test"
                action="javascript:alert('soumission de ' +
document.getElementById('champ-texte').value)"
                style="margin-left: 50px; margin-top:20px">
            <input type="text" name="champ-texte" id="champ-texte" size="20" />
            <input type="submit" id="bouton-submit">
        </form>
    </body>
</html>
```

Le body de la page ne changera pas de tout l'article. Nous nous contenterons d'y connecter les scripts nécessaires. Ainsi, pour un utilisateur n'ayant pas activé Javascript (un peu près 10% des internautes), le formulaire apparaîtra comme un formulaire normal, sans aide à la complétion.

1-B-2 - L'objet xmlHttpRequest

Le premier script que nous allons mettre en place est celui permettant de créer un objet xmlHttpRequest (ou XHR pour les intimes). Cet objet va nous permettre d'effectuer des requêtes vers notre serveur, sans avoir à recharger entièrement la page. Pour plus d'informations sur l'objet et ces méthodes, consultez l'article de siddh sur le sujet.

Pour notre part, nous allons utiliser la méthode suivante permettant de créer un nouvel objet, compatible entre tous les navigateurs actuels supportant l'objet xmlHttpRequest:

```
// retourne un objet xmlHttpRequest.
// méthode compatible entre tous les navigateurs (IE/Firefox/Opera)
function getXMLHTTP(){
  var xhr=null;
  if(window.XMLHttpRequest) // Firefox et autres
  xhr = new XMLHttpRequest();
  else if(window.ActiveXObject){ // Internet Explorer
    try {
      xhr = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (e) {
      try {
        xhr = new ActiveXObject("Microsoft.XMLHTTP");
      } catch (e1) {
        xhr = null;
      }
    }
  }
  else { // XMLHttpRequest non supporté par le navigateur
    alert("Votre navigateur ne supporte pas les objets XMLHTTPRequest...");
  }
  return xhr;
```

```
}
```

1-B-3 - Mise en place des constantes

Pour fonctionner et se mettre en place, notre script va avoir besoin de 3 constantes:

- *_documentForm* : le formulaire contenant notre champ texte
- *_inputField* : le champ texte lui-même
- *_submitButton* : le bouton submit de notre formulaire

La fonction d'initialisation de notre script prendra donc comme argument une référence sur ces trois éléments.

```
var _documentForm=null; // le formulaire contenant notre champ texte
var _inputField=null; // le champ texte lui-même
var _submitButton=null; // le bouton submit de notre formulaire

function initAutoComplete(form,field,submit){
  _documentForm=form;
  _inputField=field;
  _submitButton=submit;
  _inputField.autocomplete="off";
}
```

On désactive l'aide à la saisie des navigateurs en mettant l'attribut autocomplete de notre champ texte à *false*.

Cette fonction sera liée à l'évènement window.onload de notre page. Elle sera enrichie au fur et à mesure par des besoins supplémentaires d'initialisation que nous rencontrerons tout au long de cet article.

```
<script type="text/javascript">
window.onload = function(){initAutoComplete(document.getElementById('form-test'),
        document.getElementById('champ-texte'),document.getElementById('bouton-submit'))};
</script>
```

1-B-4 - Vérifier les changements du champ texte

Pour mettre en place notre mécanisme d'auto-complétion, nous devons être capable de détecter les changements dans le champ texte. Cela ne peut pas se faire avec le listener onchange du champ texte, car celui-ci n'est déclenché qu'au moment ou le champ perds son focus.

Détecter les changements à chaque touche appuyée peut se révéler dangereux, par exemple dans le cas des copier/coller ou autre qui peuvent soit passer inaperçu, soit saturer notre pauvre moteur Ajax de demandes.

Dans ce domaine encore, nous allons prendre exemple sur Google Suggest et mettre en place une méthode qui vérifiera périodiquement les changements survenus dans le champs texte, et exécutera une requête vers le serveur si nécessaire.

```
var _oldInputFieldValue=""; // valeur précédente du champ texte
var _currentInputFieldValue=""; // valeur actuelle du champ texte
var _resultCache=new Object(); // mécanisme de cache des requêtes

// tourne en permanence pour suggérer suite à un changement du champ texte
function mainLoop(){
  _currentInputFieldValue = _inputField.value;
  if(_oldInputFieldValue!=_currentInputFieldValue){
    var valeur=escapeURI(_currentInputFieldValue);
    var suggestions=_resultCache[_currentInputFieldValue];
    if(suggestions){ // la réponse était encore dans le cache
      metsEnPlace(valeur,suggestions)
```

```
      }else{
        callSuggestions(valeur) // appel distant
      }
      _inputField.focus()
    }
    _oldInputFieldValue=_currentInputFieldValue;
    setTimeout("mainLoop()",200); // la fonction se redéclenchera dans 200 ms
    return true
  }
```

Cette méthode sera appelée la première fois dans la fonction d'initialisation de notre script. Elle contrôle à chacun de ces passages l'état du champ texte et exécute une requête vers le serveur si nécessaire.

_resultCache est un objet qui nous permettra de constituer un cache des requêtes, pour éviter de les renvoyer systématiquement (très utile, par exemple en cas de backspace).

La fonction *metsEnPlace* mettra en place dans la page nos suggestions, et la fonction *callSuggestions* exécuteras une requête, via l'objet xmlHttpRequest, vers le serveur de données.

La méthode ci-dessus parait un peu complexe pour le résultat obtenu, mais elle va subir des changements au fur et à mesure des étapes de cet article. En particulier *_currentInputFieldValue* ne sera plus initialisée à l'intérieur de la méthode mais dans les autres méthodes de gestion des évènements.

escapeURI est une méthode toute simple, permettant d'échapper les caractères spéciaux du champ texte avant d'envoyer la requête au serveur. Cette méthode se base sur des méthodes JavaScript natives des navigateurs.

```
  // échappe les caractères spéciaux
  function escapeURI(La){
    if(encodeURIComponent) {
      return encodeURIComponent(La);
    }
    if(escape) {
      return escape(La)
    }
  }
```

1-B-5 - L'appel au serveur - méthode callSuggestions

Cette méthode va contacter notre serveur pour récupérer au plus 10 suggestions pour le texte entré dans notre champ texte.

```
  var _xmlHttp = null; //l'objet xmlHttpRequest utilisé pour contacter le serveur
  var _adresseRecherche = "options.php" //l'adresse à interroger pour trouver les suggestions

  function callSuggestions(valeur){
    if(_xmlHttp&&_xmlHttp.readyState!=0){
      _xmlHttp.abort()
    }
    _xmlHttp=getXMLHTTP();
    if(_xmlHttp){
      //appel à l'url distante
      _xmlHttp.open("GET",_adresseRecherche+"?debut="+valeur,true);
      _xmlHttp.onreadystatechange=function() {
        if(_xmlHttp.readyState==4&&_xmlHttp.responseXML) {
          var liste = traiteXmlSuggestions(_xmlHttp.responseXML)
          cacheResults(valeur,liste)
          metsEnPlace(valeur,liste)
        }
      };
      // envoi de la requête
      _xmlHttp.send(null)
    }
  }
```

Cette fonction utilise l'objet xmlHttpRequest en mode asynchrone (le premier A de Ajax), comme le montre le troisième paramètre de la méthode *open()* qui vaut *true*.

Lorsque la réponse du serveur reviendra, le listener lié à l'évènement *readyStateChange* sera déclenché. Concrètement, cela signifie que quand la requête sera terminée (readyState==4), la fonction *traiteXmlSuggestions* transformera notre document XML en une liste de suggestions (Array de string) et la méthode *metsEnPlace* sera déclenchée, avec comme argument le texte dans le champ utilisateur et les possibilités de complétion. La méthode *cacheResults* permet de garder les demandes précédentes faites au serveur.

```
// Mécanisme de caching des réponses
function cacheResults(debut,suggestions){
  _resultCache[debut]=suggestions
}
```

1-B-6 - Transformation du document XML - méthode traiteXmlSuggestions

Cette méthode va gérer la transformation de la réponse du serveur au format XML en une liste de suggestions, sous forme de tableau de chaînes de caractères.

```
// Transformation XML en tableau
function traiteXmlSuggestions(xmlDoc) {
  var options = xmlDoc.getElementsByTagName('option');
  var optionsListe = new Array();
  for (var i=0; i < options.length; ++i) {
    optionsListe.push(options[i].firstChild.data);
  }
  return optionsListe;
}
```

1-B-7 - La mise en place des suggestions - méthode metsEnPlace

La dernière fonction que nous avons à mettre en oeuvre dans cette phase est la fonction metsEnPlace, qui va mettre les suggestions en place. Ces suggestions sont mises en place dans une liste de suggestions (liste à puce UL pour le moment), créée au moment de l'initialisation de notre script d'auto-complétion.

```
var _completeListe=null; // la liste contenant les suggestions

// création d'une liste pour les suggestions
// méthode appelée à l'initialisation
function creeAutocompletionListe(){
  _completeListe=document.createElement("UL");
  _completeListe.id="completeListe";
  document.body.appendChild(_completeListe);
}

function metsEnPlace(valeur, liste) {
  while(_completeListe.childNodes.length>0) {
    _completeListe.removeChild(_completeListe.childNodes[0]);
  }
  for (var i=0; i < liste.length; ++i) {
    var nouveauElmt = document.createElement("LI")
    nouveauElmt.innerHTML = liste[i]
    _completeListe.appendChild(nouveauElmt)
  }
}
```

Dans un premier temps, la présentation sera inexistante, l'interactivité également. Ce sera l'objet des deux prochaines parties que d'améliorer la présentation de la chose et de mettre toute l'interactivité en place.

Nous en avons maintenant terminé avec la première étape de cet article. Notre page HTML est maintenant capable de dialoguer avec notre serveur (qui est ici résumé à une simple page PHP), et d'intégrer dans la page le retour de ce serveur.

Vous pouvez tester le résultat.

Cette étape représente le coeur d'une page utilisant Ajax. Mais ce n'est pas l'étape la plus compliquée. En effet, il nous reste maintenant à présenter un peu plus convivialement notre liste de suggestions et à lier les évènements survenants sur-le-champ texte à notre liste de suggestions, pour reconstruire un comportement de type liste déroulante. Ces deux étapes sont plus "ingrates" que la première étape: Il n'est plus question d'Ajax dans ces deux étapes, simplement de JavaScript habituel, avec tous les problèmes de compatibilité entre navigateurs.

Ces deux étapes sont les objets des deux sections suivantes.

2 - Etape 2 - Présentation

La seconde étape de cet article est la présentation. En effet, dans la partie précédente, les suggestions étaient intégrées dans la page sous forme de liste à puce. Si cela permet de prouver que l'insertion a bien lieu, ce n'est pas la présentation que nous cherchons à obtenir.

Pour des raisons pratiques, notre liste de suggestion se présentera sous la forme d'un div contenant un couple div/span pour chacune des suggestions.

2-A - Initialisation des règles de style

La première chose à faire est d'initialiser l'ensemble des styles que nous allons utiliser. Pour cela, nous commençons par créer la méthode *insereCSS* qui nous permet d'insérer des règles de style dans la feuille CSS, de façon indépendante du navigateur. Cela implique bien entendu qu'une feuille de style soit liée à notre document.

```
//insère une règle avec son nom
function insereCSS(nom,regle){
  if (document.styleSheets) {
    var I=document.styleSheets[0];
    if(I.addRule){ // méthode IE
      I.addRule(nom,regle)
    }else if(I.insertRule){ // méthode DOM
      I.insertRule(nom+" { "+regle+" }",I.cssRules.length)
    }
  }
}
```

La fonction ci-dessus ne fonctionne pas dans Opera (elle n'a aucun effet). En effet, le tableau styleSheets n'existe pas dans Opera. La présentation de notre liste sera donc partielle, pour cette raison dans Opera.

Nous pouvons maintenant initialiser les règles de style dont nous nous servirons par la suite:

```
function initStyle(){
  var AutoCompleteDivListeStyle="font-size: 13px; font-family: arial,sans-serif;
word-wrap:break-word; ";
  var AutoCompleteDivStyle="display: block; padding-left: 3; padding-right: 3; height: 16px;
overflow: hidden; background-color: white;";
  var AutoCompleteDivActStyle="background-color: #3366cc; color: white ! important; ";
  insereCSS(".AutoCompleteDivListeStyle",AutoCompleteDivListeStyle);
  insereCSS(".AutoCompleteDiv",AutoCompleteDivStyle);
  insereCSS(".AutoCompleteDivAct",AutoCompleteDivActStyle);
}
```

Le style AutoCompleteDivListeStyle est le style qui sera appliqué au div contenant l'ensemble des suggestions. AutoCompleteDiv est le style des div de suggestion par défaut et AutoCompleteDivAct est le style de la div de suggestion actuellement en surbrillance (cf partie 3-B).

La fonction *setStylePourElement* va nous permettre d'appliquer une classe de style à un élément:

```
function setStylePourElement(c,name){
  c.className=name;
}
```

Dans la 'vraie' version de Google Suggest, cette fonction est bien plus élaborée et permet en particulier d'appliquer un style à l'élément même sous Opera, bien que les nouvelles classes de style n'ait pas pu être intégrée par Opera. Par soucis de clarté toute cette partie a été mise de coté.

2-B - Calcul dynamique du positionnement

Dans un second temps, nous allons mettre en place les méthodes permettant de gérer dynamique le placement de notre liste de suggestions. Comme il est naturel pour une liste de suggestions celles-ci doivent apparaître directement en dessous du champ texte que l'utilisateur est en train de remplir.

La méthode suivante permet d'obtenir la position absolue dans la page d'un élément:

```
// calcule le décalage à gauche
function calculateOffsetLeft(r){
  return calculateOffset(r,"offsetLeft")
}

// calcule le décalage vertical
function calculateOffsetTop(r){
  return calculateOffset(r,"offsetTop")
}

function calculateOffset(r,attr){
  var kb=0;
  while(r){
    kb+=r[attr];
    r=r.offsetParent
  }
  return kb
}
```

La méthode suivante permet de déterminer la largeur à donner à notre liste de suggestions, qui doit être la même que pour notre champ texte. L'ajustement est lié à la largeur du bord de notre champ texte.

```
// calcule la largeur du champ
function calculateWidth(){
  return _inputField.offsetWidth-2*1
}
```

Enfin, cette dernière fonction utilise l'ensemble des méthodes précédentes pour appliquer le positionnement correct à notre liste de suggestions (qui a déjà été renommée en _conpleteDiv):

```
function setCompleteDivSize(){
  if(_completeDiv){
    _completeDiv.style.left=calculateOffsetLeft(_inputField)+"px";
    _completeDiv.style.top=calculateOffsetTop(_inputField)+_inputField.offsetHeight-1+"px";
    _completeDiv.style.width=calculateWidth()+"px"
  }
}
```

2-C - Initialisation de la div de suggestions

Nous allons maintenant remplacer la méthode *creeAutocompletionListe* que nous avions écrite dans la première partie, par la méthode *creeAutocompletionDiv*, qui prendre en compte tout ce que nous venons de mettre en place

```
var _completeDiv = null; // la div contenant la liste de suggestions

function creeAutocompletionDiv() {
  initStyle();
  _completeDiv=document.createElement("DIV");
  _completeDiv.id="completeDiv";
  var borderLeftRight=1;
  var borderTopBottom=1;
  _completeDiv.style.borderRight="black "+borderLeftRight+"px solid";
  _completeDiv.style.borderLeft="black "+borderLeftRight+"px solid";
  _completeDiv.style.borderTop="black "+borderTopBottom+"px solid";
  _completeDiv.style.borderBottom="black "+borderTopBottom+"px solid";
```

```
    _completeDiv.style.zIndex="1";
    _completeDiv.style.paddingRight="0";
    _completeDiv.style.paddingLeft="0";
    _completeDiv.style.paddingTop="0";
    _completeDiv.style.paddingBottom="0";
    setCompleteDivSize();
    _completeDiv.style.visibility="visible";
    _completeDiv.style.position="absolute";
    _completeDiv.style.backgroundColor="white";
    document.body.appendChild(_completeDiv);
    setStylePourElement(_completeDiv,"AutoCompleteDivListeStyle");
  }
```

Notre div est maintenant crée. Elle est prête à recevoir des éléments qui prendront la forme de span contenu dans des div, eux même rajoutés à la div que nous venons de créer.

2-D - Mise en place de la liste de suggestions

La dernière étape dans la présentation est la mise en place des suggestions. La fonction *metsEnPlace* de la partie 1 est complétement réécrite pour construire des éléments div/span à la place des lis. Dans notre cas, l'élément span n'a pas une utilité énorme. C'est plus un artefact de lié au nombre de hits dans Google Suggest. Mais il permet de bien se rendre compte qu'on peut rajouter des éléments de présentation dans notre liste de suggestions indépendamment des valeurs de complétion.

```
  function metsEnPlace(valeur, liste){
    while(_completeDiv.childNodes.length>0) {
      _completeDiv.removeChild(_completeDiv.childNodes[0]);
    }
    // mise en place des suggestions
    for(var f=0; f<liste.length; ++f){
      var nouveauDiv=document.createElement("DIV");
      setStylePourElement(nouveauDiv,"AutoCompleteDiv");
      var nouveauSpan=document.createElement("SPAN");
      nouveauSpan.innerHTML=liste[f]; // le texte de la suggestion
      nouveauDiv.appendChild(nouveauSpan);
      _completeDiv.appendChild(nouveauDiv)
    }
    if(_completeDivRows>0) {
      _completeDiv.height=16*_completeDivRows+4;
    } else {
      hideCompleteDiv();
    }
  }
```

Vous pouvez maintenant retester notre page, dont la présentation a bien évoluée.

3 - Partie 3 - Gestion des évènements

La dernière partie de cet article s'attaque à la gestion des évènements. En effet, notre auto-complétion a maintenant l'aspect d'une liste déroulante, mais pas encore du tout le comportement. Il va nous falloir intercepter les divers évènements, intervenant un peu partout dans le document, pour qu'on puisse par exemple passer d'une suggestion à l'autre à l'aide des touches haut/bas, qu'on puisse sélectionner une suggestion à l'aide de la touche tabulation ou au click de la souris ou pour qu'apparaisse dans le champ la possibilité de complétion.

Tous ces petits détails, qui vont faire que notre page sera réellement conviviale pour l'utilisateur, et qu'il saura l'utiliser de la façon la plus intuitive possible.

3-A - Gestion des touches

3-A-1 - Sur le document

Le premier gestionnaire d'avènement que nous allons mettre en place est un gestionnaire d'avènement qui intercepte tous les évènements clavier ayant lieu sur le onkeydown de notre document. Cette méthode se contente de stocker dans la variable *_lastKeyCode* le code de la touche à l'origine de l'évènement.

```
var _lastKeyCode=null;

// Handler pour le keydown du document
var onKeyDownHandler=function(event){
  // accès evenement compatible IE/Firefox
  if(!event&&window.event) {
    event=window.event;
  }
  // on enregistre la touche ayant déclenché l'évènement
  if(event) {
    _lastKeyCode=event.keyCode;
  }
}
```

Cette méthode est liée à l'évènement onkeydown du document, lors du chargement du document.

Elle nous servira lorsqu'un évènement fera suite à l'appui d'une touche clavier, pour pouvoir retrouver le code de la touche pressée.

```
document.onkeydown=keyDownHandler;
```

3-A-2 - Sur le champ texte

Nous allons maintenant mettre en place un gestionnaire d'évènement sur le onkeyup de notre champ texte. De cette façon, nous pourrons accéder à tous les évènements clavier intervenant sur notre champ texte et en changer éventuellement les effets.

```
var _eventKeycode = null;

// Handler pour le keyup du champ texte
var onKeyUpHandler=function(event){
  // accès evenement compatible IE/Firefox
  if(!event&&window.event) {
    event=window.event;
  }
  _eventKeycode=event.keyCode;
  // Dans les cas touches touche haute(38) ou touche basse (40)
  if(_eventKeycode==40||_eventKeycode==38) {
```

```
      // on autorise le blur du champ (traitement dans onblur)
      blurThenGetFocus();
    }
    // taille de la selection
    var N=rangeSize(_inputField);
    // taille du texte avant la sélection (sélection = suggestion d'autocomplétion)
    var v=beforeRangeSize(_inputField);
    // contenu du champ texte
    var V=_inputField.value;
    if(_eventKeycode!=0){
      if(N>0&&v!=-1) {
        // on recupere uniquement le champ texte tapé par l'utilisateur
        V=V.substring(0,v);
      }
      // 13 = touche entrée
      if(_eventKeycode==13||_eventKeycode==3){
        var d=_inputField;
        // on mets en place l'ensemble du champ texte en repoussant la sélection
        if(_inputField.createTextRange){
          var t=_inputField.createTextRange();
          t.moveStart("character",_inputField.value.length);
          _inputField.select()
        } else if (d.setSelectionRange){
          _inputField.setSelectionRange(_inputField.value.length,_inputField.value.length)
        }
      } else {
        // si on a pas pu agrandir le champ non sélectionné, on le mets en place violemment.
        if(_inputField.value!=V) {
          _inputField.value=V
        }
      }
    }
    // si la touche n'est ni haut, ni bas, on stocke la valeur utilisateur du champ
    if(_eventKeycode!=40&&_eventKeycode!=38) {
          _currentInputFieldValue=V;
    }
    if(handleCursorUpDownEnter(_eventKeycode)&&_eventKeycode!=0) {
      // si on a pressé une touche autre que haut/bas/enter
      PressAction();
    }
  }
```

Ce gestionnaire est rattaché à l'évènement onkeyup à l'initialisation de la page:

```
  _inputField.onkeyup=onKeyUpHandler;
```

Ce gestionnaire d'évènement intercepte toutes les touches pressées sur notre champ texte. Il effectue des actions spécifiques en pour les touches haut, bas et enter. Les touches haut et bas vont permettre de choisir la suggestion dans la liste et la touche enter permettra de valider le formulaire.

Pour les autres touches, il suit la partie de notre champ texte qui est entrée par l'utilisateur, en mettant à jour la variable _currentInputFieldValue et passe la gestion à la fonction *PressAction*.

L'étape de séparation de l'entrée utilisateur de la suggestion est très importante car nous devons à tout moment connaître ce que l'utilisateur a rentré indépendamment de la valeur complète de notre champ texte, qui va être une des suggestions. Ce suivi remplace complètement la mise à jour de _currentInputFieldValue que nous avions mis en place dans la partie 1-B-4.

Un certain nombre d'autres fonctions apparaissent dans cette méthode. Ce sont les méthodes:

- rangeSize : recherche la taille du texte sélectionné dans un champ
- beforeRangeSize : recherche la taille du texte placé avant la sélection dans un champ
- handleCursorUpDownEnter : gère les touches haut/bas et enter
- PressAction : gestionnaire de touches

Les deux premières méthodes (rangeSize et beforeRangeSize) traitent de la taille de la sélection dans notre champ texte. La sélection est remplie automatiquement par notre suggestion, au moment de la mise en place des possibilités. Nous verrons l'ensemble de ces mécanismes dans les parties suivantes.

Pour l'instant, tout ce que nous devons savoir est que le champ texte se compose de deux parties: le champ entré par l'utilisateur suivi d'une partie sélectionnée qui est la suggestion la plus probable.

La fonction handleCursorUpDownEnter traite spécifiquement les touches haut/bas et enter. Les touches haut/bas permettent de changer le texte actuellement sélectionné dans notre menu déroulant, la touche enter valide le formulaire avec la suggestion que nous y avons placé.

```
// Change la suggestion sélectionnée.
// cette méthode traite les touches haut, bas et enter
function handleCursorUpDownEnter(eventCode){
  if(eventCode==40){
    highlightNewValue(_highlightedSuggestionIndex+1);
    return false
  }else if(eventCode==38){
    highlightNewValue(_highlightedSuggestionIndex-1);
    return false
  }else if(eventCode==13||eventCode==3){
    return false
  }
  return true
}
```

Les touches haut/bas et enter sont capturés dans cet méthode. La méthode *highlightNewValue* sélectionne une suggestion repérée par son index. Cette méthode sera explicitée dans la prochaine partie.

La dernière méthode qui nous reste à définir dans le cadre de la gestion des touches clavier est la méthode *PressAction*. Cette méthode gère l'appui sur toutes les autres touches que haut/bas et enter. Cette méthode assure la sélection de la suggestion correcte dans la liste des suggestions quand un caractère utilisateur est ajouté. Elle est vue en détail dans la prochaine partie.

3-B - Gestion de la suggestion en surbrillance

Typiquement, dans une liste de suggestions, une des suggestions est en surbrillance. De plus, nous voulons également que dans notre champ texte apparaisse la première possibilité de complétion, en faisant en sorte que la partie générée par notre moteur soit sélectionnée pour être remplacée par une touche utilisateur si nécessaire.

3-B-1 - choix d'une suggestion en surbrillance

La première méthode à mettre en place est la méthode *PressAction*, que nous étions en train de mettre en place dans la partie précédente. Cette méthode a pour charge de trouver la nouvelle suggestion correspondant à la chaîne entrée par l'utilisateur, de la mettre en surbrillance, et de compléter le champ texte par cette suggestion.

```
var _completeDivRows = 0;
var _completeDivDivList = null;
var _highlightedSuggestionIndex = -1;
var _highlightedSuggestionDiv = null;

// gère une touche pressée autre que haut/bas/enter
function PressAction(){
  _highlightedSuggestionIndex=-1;
  var suggestionList=_completeDiv.getElementsByTagName("div");
  var suggestionLongueur=suggestionList.length;
  // on stocke les valeurs précédentes
  // nombre de possibilités de complétion
  _completeDivRows=suggestionLongueur;
```

```
    // possiblités de complétion
    _completeDivDivList=suggestionList;
    // si le champ est vide, on cache les propositions de complétion
    if(_currentInputFieldValue==""||suggestionLongueur==0){
      hideCompleteDiv()
    }else{
      showCompleteDiv()
    }
    var trouve=false;
    // si on a du texte sur lequel travailler
    if(_currentInputFieldValue.length>0){
      var indice;
      // T vaut true si on a dans la liste de suggestions un mot commencant comme l'entrée utilisateur
      for(indice=0; indice<suggestionLongueur; indice++){
        if(getSuggestion(suggestionList.item(indice)).toUpperCase().
                          indexOf(_currentInputFieldValue.toUpperCase())==0) {
          trouve=true;
          break
        }
      }
    }
    // on désélectionne toutes les suggestions
    for(var i=0; i<suggestionLongueur; i++) {
      setStylePourElement(suggestionList.item(i),"AutoCompleteDiv");
    }
    // si l'entrée utilisateur (n) est le début d'une suggestion (n-1) on sélectionne cette suggestion
  avant de continuer
    if(trouve){
      _highlightedSuggestionIndex=indice;
      _highlightedSuggestionDiv=suggestionList.item(_highlightedSuggestionIndex);
    }else{
      _highlightedSuggestionIndex=-1;
      _highlightedSuggestionDiv=null
    }
    var supprSelection=false;
    switch(_eventKeycode){
      // cursor left, cursor right, page up, page down, others??
      case 8:
      case 33:
      case 34:
      case 35:
      case 35:
      case 36:
      case 37:
      case 39:
      case 45:
      case 46:
        // on supprime la suggestion du texte utilisateur
        supprSelection=true;
        break;
      default:
        break
    }
    // si on a une suggestion (n-1) sélectionnée
    if(!supprSelection&&_highlightedSuggestionDiv){
      setStylePourElement(_highlightedSuggestionDiv,"AutoCompleteDivAct");
      var z;
      if(trouve) {
        z=getSuggestion(_highlightedSuggestionDiv).substr(0);
      } else {
        z=_currentInputFieldValue;
      }
      if(z!=_inputField.value){
        if(_inputField.value!=_currentInputFieldValue) {
          return;
        }
        // si on peut créer des range dans le document
        if(_inputField.createTextRange||_inputField.setSelectionRange) {
          _inputField.value=z;
        }
        // on sélectionne la fin de la suggestion
        if(_inputField.createTextRange){
          var t=_inputField.createTextRange();
          t.moveStart("character",_currentInputFieldValue.length);
          t.select()
        }else if(_inputField.setSelectionRange){
          _inputField.setSelectionRange(_currentInputFieldValue.length,_inputField.value.length)
        }
      }
```

```
    }else{
      // sinon, plus aucune suggestion de sélectionnée
      _highlightedSuggestionIndex=-1;
    }
  }
```

Cette méthode est une des plus longue et complexe de ce script d'auto-complétion, car elle est centrale dans notre mécanisme. Son exécution est la suivante:

1 Suppression de l'index de la suggestion précédemment sélectionnée, et recopie du div contenant les suggestions dans la variable globale _completeDivDivList.
2 Si le champ est vide ou qu'on n'a aucune suggestion, on cache les possibilités de suggestions, sinon, on les affiche.
3 On recherche si le texte utilisateur entré correspond à un début de suggestion (_currentInputFieldValue contient l'entrée utilisateur).
4 On désélectionne toutes les suggestions et éventuellement on resélectionne celle qui correspond à notre début de texte.
5 On filtre les touches spéciales (comme page up/page down et autres) qui ont pour effet d'arrêter les suggestions.
6 Si on n'est pas dans le cas d'une touche spéciale et qu'on a trouvé une suggestion possible, on met en place dans le champ texte cette suggestion en sélectionnant la partie suggérée.

3-B-2 - méthodes de gestion de la surbrillance

Nous avons utilisé dans les fonctions précédentes un certain nombre de fonctions pour la gestion de la surbrillance et des sélections dans les champs textes que nous n'avons pas encore mises en place.

```
  var _cursorUpDownPressed = null;

  // permet le blur du champ texte après que la touche haut/bas ai été pressé.
  // le focus est récupéré après traitement (via le timeout).
  function blurThenGetFocus(){
    _cursorUpDownPressed=true;
    _inputField.blur();
    setTimeout("_inputField.focus();",10);
    return
  }
```

La méthode *blurThenGetFocus* est déclenchée après l'appui sur la touche haut/bas. Elle effectue un blur sur notre champ texte, pour que l'utilisateur voie bien le changement de suggestion dans la liste des suggestions et récupère le focus pour notre champ texte.

```
  // taille de la sélection dans le champ input
  function rangeSize(n){
    var N=-1;
    if(n.createTextRange){
      var fa=document.selection.createRange().duplicate();
      N=fa.text.length
    }else if(n.setSelectionRange){
      N=n.selectionEnd-n.selectionStart
    }
    return N
  }

  // taille du champ input non sélectionne
  function beforeRangeSize(n){
    var v=0;
    if(n.createTextRange){
      var fa=document.selection.createRange().duplicate();
      fa.moveEnd("textedit",1);
      v=n.value.length-fa.text.length
    }else if(n.setSelectionRange){
      v=n.selectionStart
```

```
    }else{
      v=-1
    }
    return v
  }

  // Place le curseur à la fin du champ
  function cursorAfterValue(n){
    if(n.createTextRange){
      var t=n.createTextRange();
      t.moveStart("character",n.value.length);
      t.select()
    } else if(n.setSelectionRange) {
      n.setSelectionRange(n.value.length,n.value.length)
    }
  }
```

Les trois méthodes *rangeSize*, *beforeRangeSize* et *cursorAfterValue* permettent d'accéder, de façon indépendante du navigateur, respectivement à la longueur de la sélection dans un champ, la longueur de texte avant la sélection (qui correspond dans notre cas à la longueur de l'entrée utilisateur) et de placer le curseur tout à la fin du champ texte en créant une sélection de longueur nulle à la fin du champ.

```
  // Retourne la valeur de la possibilité (texte) contenu dans une div de possibilité
  function getSuggestion(uneDiv){
    if(!uneDiv) {
      return null;
    }
    return trimCR(uneDiv.getElementsByTagName('span')[0].firstChild.data)
  }

  // supprime les caractères retour chariot et line feed d'une chaîne de caractères
  function trimCR(chaine){
    for(var f=0,nChaine="",zb="\n\r"; f<chaine.length; f++) {
      if (zb.indexOf(chaine.charAt(f))==-1) {
        nChaine+=chaine.charAt(f);
      }
    }
    return nChaine
  }
```

La méthode *getSuggestion* retourne la valeur de la suggestion contenue dans un div donnée. Cette fonction utilise *trimCR* pour supprimer les sauts de lignes qui pourrait éventuellement s'être insérés dans les suggestions au moment du dialogue XML.

```
  // Cache complètement les choix de complétion
  function hideCompleteDiv(){
    _completeDiv.style.visibility="hidden"
  }

  // Rends les choix de completion visibles
  function showCompleteDiv(){
    _completeDiv.style.visibility="visible";
    setCompleteDivSize()
  }
```

Les fonctions *hideCompleteDiv* et *showCompleteDiv* permettent d'afficher ou de cacher la liste des complétions possibles.

```
  // Change la suggestion en surbrillance
  function highlightNewValue(C){
    if(!_completeDivDivList||_completeDivRows<=0) {
      return;
    }
    showCompleteDiv();
    if(C>=_completeDivRows){
      C=_completeDivRows-1
    }
    if(_highlightedSuggestionIndex!=-1&&C!=_highlightedSuggestionIndex){
      setStylePourElement(_highlightedSuggestionDiv,"AutoCompleteDiv");
```

```
      _highlightedSuggestionIndex=-1
    }
    if(C<0){
      _highlightedSuggestionIndex=-1;
      _inputField.focus();
      return
    }
    _highlightedSuggestionIndex=C;
    _highlightedSuggestionDiv=_completeDivDivList.item(C);
    setStylePourElement(_highlightedSuggestionDiv,"AutoCompleteDivAct");
    _inputField.value=getSuggestion(_highlightedSuggestionDiv);
  }
```

Enfin, la méthode *highlightNewValue* est utilisée pour mettre en surbrillance une nouvelle suggestion de la liste.

Notre travail sur la gestion des touches et de la surbrillance est suffisamment abouti pour que nous puissions jeter un oeil au résultat obtenu.

Le travail qu'il nous reste à faire est maintenant plus de l'ordre des détails.

3-C - Gestion de la souris

Une amélioration que nous pouvons encore apporter facilement est la gestion de la souris dans un choix de suggestion.

```
  // déclenchée quand on clique sur une div contenant une possibilité
  var divOnMouseDown=function(){
    _inputField.value=getSuggestion(this);
    _documentForm.submit()
  };

  // déclenchée quand on passe sur une div de possibilité. La div précédente est passée en style
  normal
  var divOnMouseOver=function(){
    if(_highlightedSuggestionDiv) {
      setStylePourElement(_highlightedSuggestionDiv,"AutoCompleteDiv");
    }
    setStylePourElement(this,"AutoCompleteDivAct")
  };

  // déclenchée quand la sourie quitte une div de possibilité. La div repasse a l'état normal
  var divOnMouseOut = function(){
    setStylePourElement(this,"AutoCompleteDiv");
  };
```

Ces méthodes sont liées au div de suggestion lors de lors création dans la méthode *metsEnPlace*.

```
      nouveauDiv.onmousedown=divOnMouseDown;
      nouveauDiv.onmouseover=divOnMouseOver;
      nouveauDiv.onmouseout=divOnMouseOut;
```

Elles font en sorte que la suggestion actuellement survolée soit mise en surbrillance et permettent à l'utilisateur de cliquer sur une des suggestions pour la sélectionner et valider le formulaire.

3-D - Détails finaux

Nous allons encore mettre en place quelques petits gestionnaires pour que la finition de notre auto-complétion soit réellement bonne.

Le premier gestionnaire est placé sur le onresize de la page. En effet, un resize de la page peut causer un

déplacement/changement de taille de notre champ texte, il faut donc que la liste de suggestions s'adapte.

```
// Handler de resize de la fenêtre
var onResizeHandler=function(event){
  // recalcule la taille des suggestions
  setCompleteDivSize();
}
```

Le deuxième et dernière gestionnaire que nous mettrons en place est sur le blur de notre champ texte. Dans ce cas, nous devons cacher les possibilités de complétion, et si la touche pressée est tab, passer directement au bouton submit du formulaire.

```
// Handler de blur sur le champ texte
var onBlurHandler=function(event){
  if(!_cursorUpDownPressed){
    // si le blur n'est pas causé par la touche haut/bas
    hideCompleteDiv();
    // Si la dernière touche pressé est tab, on passe au bouton de validation
    if(_lastKeyCode==9){
      _submitButton.focus();
      _lastKeyCode=-1
    }
  }
  _cursorUpDownPressed=false
};
```

Cette méthode réutilise les variables globales mises en place par nos gestionnaires de touches (_cursorUpDownPressed et _lastKeyCode) pour savoir si le blur est causé par un appui sur les touches haut/bas (dans ce cas, nous n'avons pas besoin de cacher les suggestions). Si ce n'est pas le cas, la liste de suggestions est cachée, et si c'est la touche tab qui a été pressée, nous mettons le focus directement sur le bouton submit du formulaire.

Nous pouvons maintenant consulter la dernière version de notre auto-complétion. On voit que cela se rapproche énormément de l'interface mise en place par Google Suggest. La compatibilité IE6/Firefox1.5 est totale. Le script a un fonctionnement dans Opera8.5 qui est adéquat, bien qu'il n'y ait pas de mise en surbrillance, suite au problème d'insertion de style. La page est complètement transparente pour les gens n'utilisant pas JavaScript et se comporte comme un simple formulaire (la soumission du formulaire, dans cet exemple, déclenche un message javascript qui n'est bien sûr pas disponible si javascript est désactivé, mais généralement le formulaire pointe vers une nouvelle page).