

Créer une fonctionnalité drag and drop sur votre site

par Peter-Paul Koch ([Auteur](#)) Didier Mournval ([Traducteur](#))

Date de publication : 11 janvier 2010

Dernière mise à jour :

Cet article est la traduction de : **Drag and drop**.

Voici un script simple de *drag and drop*. Il fonctionne aussi bien avec le clavier qu'avec la souris.

Présentation.....	3
Utilisation.....	3
Propriétés.....	3
L'objet dragDrop.....	3
Ce qu'est le drag and drop.....	5
Les bases.....	5
Initialiser un élément.....	5
Données de positionnement de base.....	6
Gestion de la souris.....	7
Les événements.....	7
MouseDown.....	7
Mousemove.....	8
Mouseup.....	8
Gestion du clavier.....	8
Fonctionnement de base.....	8
Les événements.....	9
Initialiser le script clavier.....	10
Déplacement au clavier.....	10
Libérer l'élément.....	12
Remerciements.....	12

Présentation

Lorsque les liens "#" de l'exemple sont activés (en cliquant dessus ou avec la touche TAB), l'élément peut être déplacé avec les touches fléchées. Appuyez sur les touches ENTER ou ESC pour déposer l'élément. Vous pouvez bien sûr modifier ces touches. Je ne suis pas certain de ce que les touches de relâchement devraient être, même si ENTER et ESC semblent être des choix opportuns.

Utilisation

- 1 Copiez l'objet dragDrop et les fonctions addEventSimple et removeEventSimple ci-dessous.
- 2 Ajustez les propriétés keyHTML et keySpeed selon vos besoins (voir les explications ci-après).
- 3 Assurez-vous que tous les éléments auxquels vous voulez appliquer le *drag and drop* ont la propriété CSS position fixée à absolute ou fixed.
- 4 Ajoutez les éléments que vous souhaitez rendre déplaçables à la fonction initElement de l'objet dragDrop. Vous pouvez référencer soit un objet HTML soit une chaîne correspondant à l'id de l'élément :
dragDrop.initElement('test');
dragDrop.initElement(document.getElementById('test2'));
- 5 Le script ajoute automatiquement une classe CSS dragged à un élément en cours de déplacement. Cela peut servir à ajouter des styles CSS.
- 6 Si vous souhaitez ajouter des actions une fois l'élément relâché, ajoutez vos instructions à la fonction releaseElement.

Propriétés

Vous avez deux propriétés à définir pour initialiser l'objet.

keyHTML correspond au code HTML que doivent contenir tous les éléments déplaçables. J'ai choisi un code HTML simple : une balise <a> avec une classe CSS pour permettre un peu de mise en forme. Vous pouvez utiliser le code HTML que vous voulez, mais il est important de noter que seul un lien <a> (en dehors des éléments de formulaire) peut obtenir le focus depuis le clavier sur tous les navigateurs, or la gestion du clavier impose qu'un élément puisse obtenir le focus pour que le script fonctionne correctement.

keySpeed indique la vitesse de déplacement de l'élément au clavier, en pixels, par appui. J'ai opté pour une valeur de 10, mais je vous encourage à tester le comportement avec d'autres valeurs pour trouver celle qui vous convient.

Il y a sept autres propriétés, mais elles sont internes au script. Elles sont initialisées à undefined et les fonctions internes leur assignent des valeurs ultérieurement. En fait, j'aurais aussi bien pu me passer de ces déclarations, mais je préfère déclarer les variables en début de script.

L'objet dragDrop

Copiez cet objet et les fonctions addEventSimple et removeEventSimple sur votre page.

```
dragDrop = {  
  keyHTML: '<a href="#" class="keyLink">#</a>',  
  keySpeed: 10, // pixels per keypress event  
  initialMouseX: undefined,  
  initialMouseY: undefined,  
  startX: undefined,  
  startY: undefined,  
  dxKeys: undefined,  
  dyKeys: undefined,  
  draggedObject: undefined,  
  initElement: function (element) {  
    if (typeof element == 'string')
```

```
    element = document.getElementById(element);
    element.onmousedown = dragDrop.startDragMouse;
    element.innerHTML += dragDrop.keyHTML;
    var links = element.getElementsByTagName('a');
    var lastLink = links[links.length-1];
    lastLink.relatedElement = element;
    lastLink.onclick = dragDrop.startDragKeys;
  },
  startDragMouse: function (e) {
    dragDrop.startDrag(this);
    var evt = e || window.event;
    dragDrop.initialMouseX = evt.clientX;
    dragDrop.initialMouseY = evt.clientY;
    addEventSimple(document, 'mousemove', dragDrop.dragMouse);
    addEventSimple(document, 'mouseup', dragDrop.releaseElement);
    return false;
  },
  startDragKeys: function () {
    dragDrop.startDrag(this.relatedElement);
    dragDrop.dXKeys = dragDrop.dYKeys = 0;
    addEventSimple(document, 'keydown', dragDrop.dragKeys);
    addEventSimple(document, 'keypress', dragDrop.switchKeyEvents);
    this.blur();
    return false;
  },
  startDrag: function (obj) {
    if (dragDrop.draggedObject)
      dragDrop.releaseElement();
    dragDrop.startX = obj.offsetLeft;
    dragDrop.startY = obj.offsetTop;
    dragDrop.draggedObject = obj;
    obj.className += ' dragged';
  },
  dragMouse: function (e) {
    var evt = e || window.event;
    var dX = evt.clientX - dragDrop.initialMouseX;
    var dY = evt.clientY - dragDrop.initialMouseY;
    dragDrop.setPosition(dX, dY);
    return false;
  },
  dragKeys: function (e) {
    var evt = e || window.event;
    var key = evt.keyCode;
    switch (key) {
      case 37: // gauche
      case 63234:
        dragDrop.dXKeys -= dragDrop.keySpeed;
        break;
      case 38: // haut
      case 63232:
        dragDrop.dYKeys -= dragDrop.keySpeed;
        break;
      case 39: // droite
      case 63235:
        dragDrop.dXKeys += dragDrop.keySpeed;
        break;
      case 40: // bas
      case 63233:
        dragDrop.dYKeys += dragDrop.keySpeed;
        break;
      case 13: // Touche Entrée
      case 27: // Touche Echap.
        dragDrop.releaseElement();
        return false;
      default:
        return true;
    }
  }
  dragDrop.setPosition(dragDrop.dXKeys, dragDrop.dYKeys);
  if (evt.preventDefault)
    evt.preventDefault();
  return false;
}
```

```
,
setPosition: function (dx,dy) {
  dragDrop.draggedObject.style.left = dragDrop.startX + dx + 'px';
  dragDrop.draggedObject.style.top = dragDrop.startY + dy + 'px';
},
switchKeyEvents: function () {
  // for Opera and Safari 1.3
  removeEventSimple(document, 'keydown', dragDrop.dragKeys);
  removeEventSimple(document, 'keypress', dragDrop.switchKeyEvents);
  addEventSimple(document, 'keypress', dragDrop.dragKeys);
},
releaseElement: function() {
  removeEventSimple(document, 'mousemove', dragDrop.dragMouse);
  removeEventSimple(document, 'mouseup', dragDrop.releaseElement);
  removeEventSimple(document, 'keypress', dragDrop.dragKeys);
  removeEventSimple(document, 'keypress', dragDrop.switchKeyEvents);
  removeEventSimple(document, 'keydown', dragDrop.dragKeys);
  dragDrop.draggedObject.className = dragDrop.draggedObject.className.replace(/dragged/, '');
  dragDrop.draggedObject = null;
}
}

function addEventSimple(obj, evt, fn) {
  if (obj.addEventListener)
    obj.addEventListener(evt, fn, false);
  else if (obj.attachEvent)
    obj.attachEvent('on'+evt, fn);
}

function removeEventSimple(obj, evt, fn) {
  if (obj.removeEventListener)
    obj.removeEventListener(evt, fn, false);
  else if (obj.detachEvent)
    obj.detachEvent('on'+evt, fn);
}
```

Ce qu'est le drag and drop

Le *drag and drop* est une technique pour pouvoir déplacer un élément à l'écran. Pour pouvoir être déplacé, l'élément doit avoir sa propriété CSS position fixée à absolute ou fixed, de façon à pouvoir le déplacer en ajustant ses coordonnées (style.top et style.left).

En théorie, il serait aussi possible d'utiliser position: relative, mais cela n'est quasiment jamais utile. D'autant que cette solution nécessiterait des calculs supplémentaires qui ne sont pas compris dans le script.

Affecter les coordonnées est relativement simple. C'est plutôt déterminer les nouvelles coordonnées qui pose problème, l'essentiel du code consiste précisément à récupérer ces nouvelles coordonnées.

Enfin, il est important de prendre l'accessibilité en considération. Habituellement, le *drag and drop* est utilisé avec la souris. C'est d'ailleurs la meilleure façon de l'utiliser. Cependant, pour permettre l'utilisation de cette fonctionnalité à ceux qui n'utilisent pas de souris, nous devons intégrer la gestion du clavier.


Les bases

Revenons sur quelques notions de base.

Initialiser un élément

Chaque script de *drag and drop* commence par l'initialisation de l'élément. Cela est réalisé par la fonction (méthode) suivante :

```
initElement: function (element) {
  if (typeof element == 'string')
    element = document.getElementById(element);
  element.onmousedown = dragDrop.startDragMouse;
  element.innerHTML += dragDrop.keyHTML;
  var links = element.getElementsByTagName('a');
  var lastLink = links[links.length-1];
  lastLink.relatedElement = element;
  lastLink.onclick = dragDrop.startDragKeys;
},
```

Si la fonction reçoit une chaîne en paramètre, elle la considère comme un id. Ensuite, elle affecte un événement onmousedown à cet élément pour enclencher la gestion à la souris. Vous noterez que j'utilise la  **gestion traditionnelle des événements** afin de préserver la référence au mot clé this dans la fonction startDragMouse.

Ensuite, la fonction récupère le paramètre keyHTML déterminé par l'auteur et le rajoute à l'élément. Ce morceau de HTML contient un seul lien et comme je l'ajoute comme dernier enfant de l'élément, je suis sûr que c'est le dernier lien de l'élément sur lequel il faudra appliquer la gestion du clavier. Le script affecte un événement onclick à ce lien pour enclencher la partie de gestion au clavier du script. Le script stocke enfin une référence à l'élément dans relatedElement, nous aurons besoin de cette référence plus tard.

Maintenant, le script n'attend plus que les actions de l'utilisateur.


Données de positionnement de base

J'ai opté pour l'approche suivante concernant le positionnement : d'abord, je récupère la position initiale de l'objet HTML au moment du début du déplacement, position stockée dans startX et startY. Ensuite, le script calcule le changement de position de la souris ou le nombre de touches fléchées appuyées pour déterminer le déplacement depuis la position initiale.

Les variables startX et startY sont définies dans la fonction startDrag, utilisée aussi bien dans la gestion à la souris qu'au clavier.

```
startDrag: function (obj) {
  if (dragDrop.draggedObject)
    dragDrop.releaseElement();
  dragDrop.startX = obj.offsetLeft;
  dragDrop.startY = obj.offsetTop;
  dragDrop.draggedObject = obj;
  obj.className += ' dragged';
},
```

Tout d'abord, si un élément est déjà référencé comme "en déplacement", le déréférencer (nous reviendrons sur releaseElement plus tard).

Ensuite, la fonction détermine la position de l'élément au début du déplacement à l'aide des propriétés offsetLeft et offsetTop (voir mon article  **Find position**) et la stocke dans les variables startX et startY pour pouvoir être réutilisée ultérieurement.

Ensuite, elle stocke une référence à l'élément déplacé dans draggedObject et y ajoute la classe CSS "dragged", afin que l'auteur puisse ajouter des styles CSS à l'élément en cours de déplacement.

Il peut arriver que l'élément à déplacer soit différent de l'élément sur lequel vous cliquez ; par exemple, le déplacement peut être initié en cliquant sur un titre. Dans ce cas, assurez-vous que la variable draggedObject fasse référence à l'élément à déplacer.

Une fois que l'utilisateur déplace l'élément (à la souris ou au clavier) commence la partie compliquée du script. Le script détermine de combien doit être déplacé l'élément et stocke ces valeurs dans `dx` et `dy`. Il suffit ensuite d'ajouter ces valeurs à `startX` et `startY` pour déterminer la nouvelle position.

Le repositionnement se fait avec cette fonction :

```
setPosition: function (dx,dy) {  
    dragDrop.draggedObject.style.left = dragDrop.startX + dx + 'px';  
    dragDrop.draggedObject.style.top = dragDrop.startY + dy + 'px';  
},
```

La fonction reçoit en paramètres `dx` et `dy`, calculés à partir du script de gestion du clavier ou de la souris puis ajuste les propriétés `style.top` et `style.left`. L'élément est déplacé.

C'est assez simple à réaliser, ce qui l'est moins est d'obtenir les bonnes valeurs pour `dx` et `dy`. Les parties souris et clavier font le calcul assez différemment, nous allons donc les détailler séparément.

Gestion de la souris

Le script concernant la gestion de la souris est plus compliqué que celui du clavier pour ce qui est des calculs. En revanche, il est plus simple au niveau de la compatibilité entre navigateurs. Nous allons donc commencer par lui.

Les événements

Tout d'abord, il faut se demander quels événements seront utiles. Il est évident que nous aurons besoin de *mousedown*, *mousemove* et *mouseup*, afin de pouvoir sélectionner, déplacer, puis relâcher un élément.

Il est aussi évident que la séquence commencera par *mousedown* sur l'élément à déplacer. De ce fait, il faudra attacher un événement *onmousedown* à tous les éléments concernés pour pouvoir les préparer à être déplacés. Nous avons déjà vu le code qui s'occupe de cela dans `startDrag` :

```
element.onmousedown = dragDrop.startDragMouse;
```

En revanche, les événements *mousemove* et *mousedown* devront être attachés non plus à l'élément mais à tout le document. La raison est que l'utilisateur pourra faire des mouvements amples et rapides sur la page et relâcher l'élément aussitôt. Ainsi, si ces événements étaient liés à l'élément, il devient possible d'en perdre le contrôle si la souris bouge trop vite et ne survole plus l'élément en cours de déplacement. Il s'agit d'une mauvaise ergonomie.

En revanche, si l'on attache le *mousemove* et le *mouseup* sur l'ensemble du document, l'élément suivra la souris, même si celle sort de l'emplacement qu'il occupe. C'est une bonne (ou du moins une meilleure) ergonomie.

Enfin, il ne faut appliquer ces événements que lorsque l'élément est sélectionné et les supprimer lorsqu'il est relâché. Cela permet d'avoir un code propre et améliore les performance car *mousemove* n'est déclenché que lorsque c'est utile.

Mousedown

Lorsqu'un événement *mousedown* est déclenché sur un élément déplaçable, la fonction `startDragMouse` est exécutée :

```
startDragMouse: function (e) {  
    dragDrop.startDrag(this);
```

```
var evt = e || window.event;
dragDrop.initialMouseX = evt.clientX;
dragDrop.initialMouseY = evt.clientY;
addEventListener(document, 'mousemove', dragDrop.dragMouse);
addEventListener(document, 'mouseup', dragDrop.releaseElement);
return false;
},
```

D'abord, elle déclenche la fonction `startDrag` dont nous avons déjà parlé. Ensuite, elle récupère les coordonnées de la souris et les stocke dans `initialMouseX` et `initialMouseY`. Plus tard, nous comparerons la position de la souris avec ces valeurs.

Enfin, la fonction retourne `false`, ceci afin d'inhiber l'action par défaut d'un événement de souris, à savoir la sélection de texte. Nous ne voulons pas sélectionner aucun texte pendant le déplacement, cela serait perturbant.

Enfin, la fonction assigne les événements *mousemove* et *mouseup* au document (pour les raisons que nous avons déjà évoquées). Cependant, il est possible que la page contienne déjà des événements *mousemove* et *mouseup*, c'est pourquoi j'utilise ma fonction `addEventListener` pour éviter de les écraser.

Mousemove

Maintenant, dès que l'utilisateur bouge la souris, la fonction `dragMouse` est déclenchée.

```
dragMouse: function (e) {
    var evt = e || window.event;
    var dx = evt.clientX - dragDrop.initialMouseX;
    var dy = evt.clientY - dragDrop.initialMouseY;
    dragDrop.setPosition(dx, dy);
    return false;
},
```

Cette fonction récupère les coordonnées actuelles de la souris (`clientX` et `clientY`) et leur soustrait `initialMouseX` et `initialMouseY`. Nous obtenons ainsi les coordonnées où nous devons positionner l'élément déplacé. Nous envoyons donc ces valeurs (`dx` et `dy`) à la fonction `setPosition`.

Là encore, nous retournons `false` pour éviter la sélection de texte.

Mouseup

Lorsque le bouton de la souris est relâché, nous appelons la fonction `releaseElement`. Nous en parlerons plus tard.

Gestion du clavier


Voici maintenant la partie la plus compliquée : le script de gestion du clavier. Contrairement à la gestion de la souris, il n'existe pas (encore) d'interface standard pour gérer le *drag and drop* au clavier. Le fonctionnement de base n'est pas trop complexe, détaillons-le rapidement.

Fonctionnement de base

Les touches utilisées pour gérer le déplacement sont naturellement les touches fléchées. C'est assez simple à gérer.

La question de l'activation et de la désactivation de l'élément est plus délicate. Il s'agit d'une partie du script qui pourrait être améliorée.

J'ai choisi de déclencher le script clavier par un lien ajouté à chaque élément déplaçable. Il n'y a pas beaucoup d'autres possibilités, nous avons besoin d'un lien car c'est un des seuls éléments *HTML* qui peut recevoir le focus sur tous les navigateurs (d'accord, les éléments de formulaires le peuvent aussi et vous pourriez peut-être utiliser une case à cocher) ; de plus, placer le lien à l'intérieur de l'élément auquel il est rattaché me semble le plus logique (bien sûr, vous pourriez le placer n'importe où sur la page, mais comment savoir à quel élément le lien est rattaché).

J'ai décidé que l'élément serait libéré en appuyant sur ENTREE ou ECHAP ; en fait, j'ai choisi ces touches parce que je n'en voit pas d'autres évidentes pour le faire. Si vous optez pour d'autres touches, pensez à modifier le numéro de code ( [voir les codes des touches](#)) :

```
case 13: // Entrée
case 27: // Echap.
    dragDrop.releaseElement();
    return false;
```

Les événements

L'événement gérant l'activation est le clic. Cet événement est accessible soit par un clic de souris, soit par l'appui sur entrée quand l'élément a le focus.

Pour être précis, l'élément est d'abord activé en mode souris (mousedown), puis libéré (mouseup) puis activé en mode clavier (click).

Les autres événements utilisés sont plus confus. Les événements clavier sont assez compliqués, particulièrement pour les touches fléchées.

Le premier problème est que nous avons besoin d'un événement qui se répète ; c'est-à-dire que si l'utilisateur garde la touche enfoncée, l'événement doit être déclenché de façon répétitive, afin que l'élément continue à se déplacer. Nous utilisons l'événement keypress pour cela.

Malheureusement, IE ne déclenche pas cet événement pour les touches fléchées. Cela peut se contourner car l'événement keydown est déclenché de façon répétitive sous IE. Il semblerait que nous devions utiliser l'événement keydown.

Vous l'aurez probablement deviné, ce n'est pas aussi simple. En effet, avec Opera et Safari 1.3, l'événement keydown ne se répète pas, de ce fait, si l'utilisateur garde une touche enfoncée, rien ne se passe après le premier mouvement. Avec ces navigateurs, nous aurons donc besoin de keypress. Firefox et Safari 3 acceptent la répétition pour les deux événements, ce qui est selon moi la meilleure solution !

Donc idéalement, nous utilisons keypress, s'il n'est pas disponible, nous utilisons keydown. Seulement, comment passer d'un événement à l'autre ? Comment savoir si keypress est disponible ?

Ma solution est d'initialiser un événement keypress. Si cet événement est déclenché, c'est donc qu'il est disponible et nous pouvons intervertir les événements liés aux touches.

La fonction startDragKeys affecte les événements pour keydown et keypress :

```
addEventSimple(document, 'keydown', dragDrop.dragKeys);
addEventSimple(document, 'keypress', dragDrop.switchKeyEvents);
```

Au départ, keydown lance la fonction dragKeys qui gère le déplacement. Le premier événement est toujours déclenché et l'élément bouge toujours. Mais en l'état actuel, rien d'autre ne se passe avec Opera et Safari 1.3.

C'est pourquoi nous avons aussi besoin de keypress. Le premier événement keypress lance la fonction switchKeyEvents qui réaffecte les événements :

```
switchKeyEvents: function () {
  removeEventSimple(document, 'keydown', dragDrop.dragKeys);
  removeEventSimple(document, 'keypress', dragDrop.switchKeyEvents);
  addEventSimple(document, 'keypress', dragDrop.dragKeys);
},
```

Cette fonction supprime les gestionnaires d'événements que nous venons d'affecter et en ajoute un nouveau : dragKeys est désormais déclenché par keypress. De plus, comme cette fonction est exécutée uniquement sur les navigateurs supportant keypress, nous avons transféré les événements du clavier vers keypress uniquement pour ces navigateurs.

Initialiser le script clavier

Lorsque l'utilisateur active le lien dans le coin de l'élément déplaçable, startDragKeys est lancée.

```
startDragKeys: function () {
  dragDrop.startDrag(this.relatedElement);
  dragDrop.dXKeys = dragDrop.dYKeys = 0;
  addEventSimple(document, 'keydown', dragDrop.dragKeys);
  addEventSimple(document, 'keypress', dragDrop.switchKeyEvents);
  this.blur();
  return false;
},
```

Tout d'abord, la fonction startDrag que nous venons de voir est appelée. La variable relatedElement, créée dans initElement et contenant une référence à l'élément déplaçable, lui est transmise.

Ensuite, elle initialise dXKeys et dYKeys à 0. Ces variables enregistreront les changements de position de l'élément.

Ensuite, les gestionnaires d'événement vus précédemment sont initiés.

Ensuite apparaît une forme de correction de bug : nous retirons le focus de l'élément qui vient d'être cliqué. Ceci est nécessaire car l'appui sur la touche entrée doit libérer l'élément. Or, si nous ne retirons pas le focus et que l'utilisateur appuie sur entrée, l'élément sera bien libéré, mais juste après, un événement click sera lancé sur le lien, ce qui ramènera l'élément en mode déplacement. Il serait donc impossible de libérer l'élément avec la touche entrée. Le problème disparaît si l'on retire le focus du lien.

Enfin, la fonction retourne false afin d'inhiber le comportement par défaut du lien.

Déplacement au clavier

La fonction dragKeys gère le déplacement au clavier.

```
dragKeys: function(e) {
  var evt = e || window.event;
  var key = evt.keyCode;
```

Nous déterminons d'abord quelle touche a été pressée (voir mon article  **Detecting keystrokes**).

Ensuite, nous utilisons une instruction switch pour déterminer l'action à effectuer en fonction de la touche pressée. L'utilité de cette partie du script est de mettre à jour dXKeys et dYKeys, qui contiennent le nombre de pixels dont l'élément a été déplacé.

```
switch (key) {  
  case 37: // gauche  
    case 63234:  
      dragDrop.dXKeys -= dragDrop.keySpeed;  
      break;  
  case 38: // haut  
    case 63232:  
      dragDrop.dYKeys -= dragDrop.keySpeed;  
      break;  
  case 39: // droite  
    case 63235:  
      dragDrop.dXKeys += dragDrop.keySpeed;  
      break;  
  case 40: // bas  
    case 63233:  
      dragDrop.dYKeys += dragDrop.keySpeed;  
      break;  
}
```

L'auteur du site devra définir le nombre de pixels dont sera déplacé l'élément à chaque appui de touche dans la variable keySpeed. En fonction de la touche appuyée, on ajoute ou soustrait cette valeur de dXKeys ou dYKeys.

Vous constaterez la présence des cas 63232-63235. Ils servent à Safari 1.3 qui n'utilise pas les keyCodes usuels de 37-40 pour les touches fléchées (mais Safari 3 le fait).

```
case 13: // Entrée  
case 27: // Echap.  
  dragDrop.releaseElement();  
  return false;
```

Si l'utilisateur appuie sur ENTREE ou ECHAP, l'élément est libéré (voir ci-dessous) et la fonction se termine. Si vous souhaitez modifier les touches de relâchement, remplacez juste leur keyCode dans le code précédent.

```
default:  
  return true;  
}
```

Si l'utilisateur appuie sur n'importe quelle autre touche, son comportement par défaut est autorisé et la fonction se termine.

```
dragDrop.setPosition(dragDrop.dXKeys, dragDrop.dYKeys);
```

dXKeys ou dYKeys étant maintenant à jour, nous envoyons leurs valeurs à setPosition pour modifier la position de l'élément.

```
if (evt.preventDefault)  
  evt.preventDefault();  
return false;  
},
```

Enfin, nous inhibons le comportement par défaut des touches, c'est-à-dire que si par exemple la touche bas est appuyée, la page est sensée descendre. Nous évitons cela avec la méthode `preventDefault` du modèle d'événement conforme aux spécifications du W3C. Pour les autres (dont IE), nous retournons `false`.

Libérer l'élément

Lorsque l'utilisateur libère l'élément, la fonction `releaseElement` est appelée. Elle supprime tous les événements mis en place dans le script, supprime la classe CSS *dragged*, détruit l'objet `draggedObject` et attend une nouvelle interaction de l'utilisateur.

```
releaseElement: function() {  
    removeEventSimple(document, 'mousemove', dragDrop.dragMouse);  
    removeEventSimple(document, 'mouseup', dragDrop.releaseElement);  
    removeEventSimple(document, 'keypress', dragDrop.dragKeys);  
    removeEventSimple(document, 'keypress', dragDrop.switchKeyEvents);  
    removeEventSimple(document, 'keydown', dragDrop.dragKeys);  
    dragDrop.draggedObject.className = dragDrop.draggedObject.className.replace(/dragged/, '');  
    dragDrop.draggedObject = null;  
}
```

Vous voudrez peut-être prévoir des actions à effectuer après le déplacement, dans ce cas, vous pouvez ajouter vos propres fonctions à appeler à la fin de `releaseElement`.

Remerciements

Je tiens à remercier [koala01](#) et [jacques_jean](#) pour leur relecture attentive de cet article.