M2202 Algorithmique

TD 4 : Création d'élément à la volée et techniques Image

Nous verrons dans ce TD comment :

- Créer et modifier des éléments a la demande ;
- Gérer les images et leur chargement dans jQuery .

Créer des éléments à la volée

La création d'éléments a la volée consiste à créer des éléments html n'existant pas dans le fichier html d'origine ; ces éléments créés à la volée peuvent ainsi avoir une existence temporaire dans la page (galerie d'images type litebox) ou permanente (menus déroulants, etc.). Cette technique permet de mettre en œuvre des fonctionnalités complexes tout en minimisant la mise en place html/css de base, les éléments additionnels étant créés à la demande avec jQuery.

On utilise les capacités syntaxiques évoluées de **jQuery** ainsi que les méthodes qui lui sont propres !!

Étape 1 - créer l'élément → on utilise notre fonction de base `$()` en lui passant une chaine correspondant à la **balise** de l'élément sous forme de chaine :

```
var une_div = $('<div />') ; //la variable une_div contient un élément virtuel de type div.
```

Étape 2 - insérer l'élément dans son parent sur la page html → la méthode `appendTo()` :

```
une_div.appendTo('body'); //la div est insérée à la fin de body.
```

Notez que jQuery autorise les deux écritures en une :

```
$('<div />').appendTo('body');//une div est créée puis insérée à la fin de body sans passer par une variable.
```

jQuery permet de créer des éléments html 'complets', attributs compris :

```
var lien = $('<a href="http://www.google.fr">Le Site de Google !</a>');
```

On peut utiliser la méthode `attr()` pour ajouter des attributs à l'élément :

```
$('<img />').attr('src', 'image1.jpg') ; //affecte l'url image1.jpg à l'attribut source de l'élément image créé.
```

Les attributs sont généralement ajoutés avant d'insérer l'élément dans le flux, mais ce n'est pas toujours le cas. Nous allons voir comment, depuis jQuery 1.4, on peut affecter proprement des attributs à un élément nouvellement créé.

Passage d'attribut à un élément html

L'ajout d'attributs pour notre nouvel élément peut se faire directement lors de la création de celui-ci. Le code peut vite devenir redondant et difficile à lire si vous procédez ainsi et qu'il y a beaucoup d'attributs. Depuis **jQuery 1.4**, il est possible de donner un **objet** en second argument à la méthode principale, cet objet contenant la liste de tous les attributs du nouvel élément, à la manière d'attr(). Il est ainsi possible de donner un grand nombre d'attributs sans altérer la lisibilité du code jQuery :

```
$('<div />', {
  id : "contenu", // identifiant de l'élément
  css : { // style css de l'élément
    color : "darkblue",
    fontSize : "1.5em"
  }
});
```

Élément à la volée : cas concret

A titre d'exemple, nous allons créer un tableau html à la volée Ceci nous permettra par ailleurs de retrouver les structures algorithmiques d'itération que sont les boucles. Nous verrons ensuite comment créer un diaporama d'images simple.

Exercice 1 : Création d'un tableau html à la volée

Le but est ici de créer la **tableau html** dont les éléments constituants (**table**, **tr** et **td**) seront **créés avec jQuery**. Nous souhaitons par ailleurs pouvoir modifier facilement le nombre de colonnes et de lignes. C'est pourquoi ces deux informations sont stockées dans **2 variables**.

Insertion de la bibliothèque jQuery et début du code

Créez une page html pour y insérer le code (vous compléterez ce code au fur et à mesure des étapes) :

```
<script src='http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js'></script>
<script>
//Déclaration-affectation des variables
var nbtr = 6;//nb de tr
var nbtd = 4;//nb de td
```

var permet de déclarer une ou plusieurs variables. Placée 'a la racine' du code (et non dans une fonction), cette déclaration est globale a tout le code, y compris pour les fonctions qui réutiliserons ces variables.

Création des éléments dans la méthode ready()

Dans la mesure ou notre tableau sera ajouté à body, il est logique de le créer lorsque body a lui-même été créé !

```
$(document).ready(function(){//lorsque la page est chargée
     //Création du tableau nbtr x nbtd
     //tab est une variable qui "pointera" sur le tableau html pour le manipuler avec jQuery
     tab = $('<table />', {//objet pour les attributs
          css : {//attribut css
          border: 'blue solid 1px'
          }
     });
```

On utilise la méthode décrite précédemment. Celle-ci permet d'ajouter l'attribut css au tableau créé.

Création des tr et td

```
// on créé les tr
```

```
for(i=0;i<nbtr;i++){
	//chaque tr doit être ajouté au tableau (variable tab)
	tr = $('<tr />').appendTo(tab);//tr -> variable
	// et pour chaque tr on créé les td
	for(j=0;j<nbtd;j++){
		//chaque td doit être ajouté au tr (variable tr)
		td = $('<td />').appendTo(tr);
	}
}
```

Nous utilisons ici une **structure itérative (boucle for)** dont **l'indice i** commence à 0 et va jusqu'à `nbtr-1` par pas de 1. Ceci permet d'exécuter la boucle **nbtr** fois. A **chaque tour** de cette boucle, nous créons **un élément tr** que nous stockons dans une variable. Ceci est nécessaire, car une deuxième boucle doit ajouter les éléments td dans ce même tr !! C'est le rôle de la boucle d'indice j, qui fonctionne selon un principe similaire a la boucle d'indice i.

Ajout du tableau à body

```
//on peut ajouter les tableau html à body..
tab.appendTo('body');
```

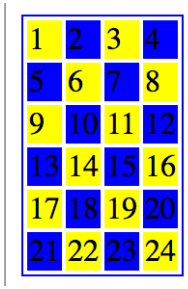
C'est à ce moment seulement qu'apparait le tableau dans la page. Notez que ce tableau aurait pu être placé de même dans un élément div ou autre. Notez aussi l'intérêt des variables, qui permettent de manipuler des éléments, même si ces éléments ne sont pas encore insérés dans le flux. Nous retrouverons ce principe de base dans le pré-chargement des images.

Quelques manipulation de style : la méthode each()

```
	//mise en forme css
	$('td').css('background-color','blue');//tout bleu
	$('tr:even td:even, tr:odd td:odd').css('background-color','yellow');
	//Mettre des valeurs croissantes
	$('td').each(function(i){//Explorer un objet-tableau jQuery avec each!!
		$(this).text(i+1);
	});
});//fin de la méthode ready()
</script>
```

On constate ici l'intérêt de la méthode `each()`, qui permet de parcourir tous les éléments d'un objet (ce qui est le cas ici) ou d'un tableau. `$(this)` correspond alors à l'élément en cours dans la boucle. Notez aussi la sélection grâce aux pseudo-classes `:even` (pair) et `:odd` (impair), qui permettent une sélection particulièrement intéressante dans notre tableau.

Le tableau final :



Exercice 2 : Diaporama interactif

Il s'agit d'afficher une série d'images dans un même conteneur. Les images sont présentes sur le site (images.zip).

Téléchargez le fichier zip et décompressez-le dans un dossier.

Créer ensuite une page html type 'jQuery' avec une div pour les images :

```
<!DOCTYPE html>
<html>
<head>
     <meta charset="utf-8">
     <script src='http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js'></script>
     <script src='diapo1.js'></script>
</head>
<body>
     <div id='conteneur_img'>
     </div>
</body>
</html>
```

Notez l'insertion d'une page JavaScript (diapo1.js), qui permet d'ajouter de grandes portions de codes sans surcharger le html (c'est aussi un procédé valide W3C).
Notre premier diaporama sera entièrement contrôlé par l'utilisateur : le changement d'image (6 images) se fera en cliquant sur l'image même.

Nous devrons donc mettre en œuvre :
– une fonction de préchargement des images ;
– l'insertion d'un élément image (dans une div)
– le changement de source au clic sur l'image

Les variables et le tableau des sources d'images

Les tableaux permettent de stocker plusieurs éléments sous le même nom de variable. Dans cet exercice, nous utiliserons un tableau pour stocker les url-sources des images.
Les variables permettent de stocker (écrire) et récupérer (lire) une information particulière. Ici, ce sont :

nb → le nombre d'images
actuel → l'image courante affichée (c'est un entier qui va de 0 a nb-1)
imgsrep → le dossier ou sont stockées les fichiers d'images
imgs → le tableau des urls sources.

Les lignes de code qui suivront seront à saisir dans le fichier diapo1.js.

```
//Déclaration de variables
var nb = 6 ; //nb d'images
var actuel = 0 ; //image courante
var imgsrep = 'images/' ; //répertoire des images
var imgs = new Array() ; //tableau des images pour préchargement
```

On a établi une convention de nommage pour toutes les images. Les fichiers seront nommés '**image' suivi d'un numéro de 0 à 5** terminés bien sûr par l'extension (ici : **.jpg**). Le **tableau** est ensuite rempli de chacune des urls (incluant le dossier *images*) dans une boucle **for**.

```
//Création du tableau des sources : le nommage permet des structures algorithmiques élégantes
//Les images sont nommées image0.jpg, image1.jpg,..
for (i=0;i<nb;i++){
     imgs.push(imgsrep + 'image' + i + '.jpg');
}
```

Notez la reconstruction de l'url dans l'expression `imgsrep + 'image' + i + '.jpg'`
En bleu : les chaines de caractères. En rouge : les variables.

Règle fondamentale en JS : dans une expression incluant le signe +, si un seul des membres est une chaine, tout est transformé en chaine (il y a alors concaténation). Les calculs peuvent bien sur être protégés par des parenthèses : `'image' + (i+1)`

Fonction de préchargement des sources

Il s'agit de précharger les images dans le cache afin d'éviter les temps d'attente lors des premiers clics. En html, le fichier est chargé dans le cache lorsque l' élément img est inséré dans le l'élément body. En réalité, c'est lorsqu'on **définit une source pour l'élément img que le fichier est chargé** dans le cache. Si l'on affecte une source à un élément img virtuel (non ajoute dans le html), le chargement s'effectuera également. C'est cette technique qui est généralement utilisée.

```
//fonction de préchargement des images
function charge_images(){
     $(imgs).each(function(){//pour chaque élément du tableau d'images -> pour chaque image
          $("<img/>").src = this; //on affecte à la source d'un img virtuel la valeur de
          l'élément(this)
     });
}
```

`$(imgs)` → transforme le tableau en un objet jQuery auquel on peut appliquer la méthode `each()`.
`$("<img/>").src = this;` → on créé un img virtuel donc on modifie la source.

Nous allons maintenant insérer dans la méthode `$(document).ready()` le code pour :

- appeler la fonction de préchargement ;
- ajouter un élément **img** a la **div** dont l'**id** a pour valeur *conteneur_img* ;
- affecter le comportement click de l'image pour en changer la source.

```
$(document).ready(function(){//lorsque la page est chargée
     //préchargement des images
     charge_images();
     //Insertion d'un élément img dans la div d'id conteneur_img
     $('<img id=\'voir\' alt=\'une image\' />').attr('src', imgsrep + 'image0.jpg').appendTo($
     ('#conteneur_img'));
     //Affectation du comportement click
     $('#voir').click(function(){
     actuel++; //on incrémente actuel
     if(actuel === nb){actuel=0;} //Test de dépassement
     $(this).attr('src', imgs[actuel]);//nouvelle source
```

```
    });
});
```

A noter : Lors de la création de l'élément image, on lui ajoute deux attributs : id et alt. L'id sera utilisé pour cibler l'élément avec jQuery lors de l'ajout du comportement. Un test unaire permet de ne pas dépasser le nombre d'image prévu : à la dernière image, on revient à la première.

Enregistrez votre fichier JS et testez votre page.

Dans le deuxième exercice, nous allons modifier le code pour que les images défilent automatiquement. Pour ce faire, nous utiliserons la fonction JavaScript `setInterval(une_fonction, intervalle)`. Cette fonction créé un objet timer qui appelle la fonction *une_fonction* tous les *intervalle* ms. Pour arrêter le timer, on appelle la fonction `clearInterval(timer)`.
Ex. :

```
t = setInterval(change_image, 1000) ; //appelle change_image toutes les secondes
clearInterval(t) ;/ /arrête l'objet timer
```

Enregistrez diapo1.js sous le nom diapo2.js et modifiez la page html pour importer ce nouveau script diapo2.js.
En bleu, les modifications a apporter :

```
//Déclaration de variables
var nb = 6;//nb d'images
var actuel = 0;//image courante
var imgsrep = 'images/'; //rép. images
var imgs = new Array();//tableau des images pour préchargement
var t; //objet timer
//Création du tableau des sources : le nommage permet des structures algorithmiques élégantes
//Les images sont nommées image0.jpg, image1.jpg, ..
for (i=0;i<nb;i++){
imgs.push(imgsrep + 'image' + i + '.jpg');
}
$(document).ready(function(){//lorsque la page est chargée
//préchargement des images
charge_images();
```

```
//Insertion d'un élément img dans la div d'id conteneur_img
$('<img id=\'voir\' alt=\'une image\' />').attr('src', imgsrep + 'image0.jpg').appendTo($('#conteneur_img'));
//Démarrage du timer
t = setInterval(change_image, 1000);
//Affectation du comportement click
$('#voir').click(function(){
clearInterval(t);//Arrêt du timer t
});
});
//fonction changement d'image
function change_image(){
actuel++; //on incrémente actuel
if(actuel===nb){actuel=0;} //Test de dépassement
$('#voir').attr('src', imgs[actuel]);//nouvelle source
}
//fonction de préchargement des images
function charge_images(){
//on créé un img jQuery virtuel dont on affecte la source -> provoque le chargement dans le cache
$(imgs).each(function(){//pour chaque élément du tableau d'images -> pour chaque image
$("<img/>").src = this; //on affecte à la source d'un img virtuel la valeur de l'élément(this)
});
}
```