SUPPLEMENT : jQuery

« Write less, do more. »

Pourquoi jQuery ?

- jQuery est un framework JavaScript, c'est-à-dire un ensemble d'outils qui vont nous simplifier l'écriture du code JavaScript :

  - Assure la compatibilité du code entre **TOUS** les navigateurs
  - Permet d'effectuer des manipulations HTML et de gérer les interactions utilisateur de façon beaucoup plus simple qu'en code natif
  - Régulièrement mis à jour

  - Remarque : tous les composants jQuery sont disponibles via la variable « $ »

Utiliser jQuery

- Toutes les fonctionnalités qu'offre jQuery peuvent être utilisé une fois que l'on a intégré le fichier .js de jQuery dans notre page
  - http://www.jquery.com/

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

La syntaxe de selection

- Grace à jQuery, nous pouvons utiliser une syntaxe similaire aux sélecteurs CSS pour récupérer les éléments DOM de notre page.
- Exemples :

  - $('#conteneur p');
    - Récupère le ou les objets DOM « p » contenus dans un élément ayant pour ID « conteneur »

  - $('table#articles td.prix');
    - Récupère le ou les objets DOM « td » ayant pour classe « prix » et contenus dans un élément « table » ayant pour ID « articles »

Gestion des évènements

- La gestion des évènements en jQuery suit les même principes que la gestion des évènements en JavaScript, mais de façon plus raccourcie :

```
$('#navigation button').on('click', function(event)
{
    console.log('Click sur le bouton de
navigation');
});
```

- Avantage : jQuery s'occupe de la compatibilité de notre code avec les navigateurs plus anciens

Gestion des évènements : exemples

- Il existe quelques nuances de récupérations des informations en jQuery :

```
$('#formulaireInscription input').on('change', function(event)
{
    console.log($(this).val()); // Affiche le texte entré dans
l'input
});

$('#contenu p').on('mouseover', function(event) {
    $(this).text('Texte au survol de la sourie.');
    // Modifie le texte contenu dans la ou les balises « p »
contenues dans la balise ayant pour ID « contenu »
});
```

Appel à des données distantes

- Une requête AJAX avec jQuery s'écrit comme suit :

```
var xhr = $.ajax({
   url: 'getArticles.php',
   type: 'GET',
   data: {page: 2}
});
xhr.done(function (data) {
   console.log('success');
});
xhr.fail(function (xhr, status) {
   console.log('error');
});
xhr.always(function () {
   console.log('complete');
```

Appel à des données distantes

- La requête AJAX est initialisée et envoyée dès l'appel à $.ajax()
- Une fois de plus, jQuery assure la compatibilité de la requête AJAX avec tous les navigateurs

- Il existe de nombreux autres paramètres pour les requêtes AJAX avec jQuery (cache, contentType, timeout, …)

Assurer l'expérience utilisateur

- Ne JAMAIS envoyer de requête AJAX synchrone : le navigateur se retrouve paralysé en attendant la réponse du serveur

- Une requête AJAX n'étant pas considérée comme un chargement de page classique, l'historique du navigateur n'évolue pas : cela peut être résolu grâce à l'Api History d'HTML5

SUPPLEMENT : La gestion de l'historique

History API

- Normalement, chaque URL doit correspondre à une ressource (document html, image, etc.)
- Le fait de dynamiser son contenu avec l'AJAX brise cette cohérence.
- L'API History permet de rétablir depuis JavaScript le lien entre l'URL et le contenu de la page.

History API

- Se déplacer dans l'historique de navigation:

```
window.history.back(); // simule un
clic sur le bouton « retour » du
navigateur

window.history.forward(); // simule un
clic sur le bouton « avancer » du
navigateur

window.history.go(-2); // recule de 2
pages

window.history.length; // nombre de
```

History API

- Ajouter un « état » à l'historique de navigation:

```
window.history.pushState(data, title,
url);
```

/!\ Le navigateur va utiliser la nouvelle URL, mais ne va pas déclencher de nouvelle requête HTTP !

```
window.history.replaceState(data,
title, url);
```

La même chose que pushState, mais modifie l'état courant au lieu d'en créer un nouveau

History API

- L'évènement « popstate » permet de réagir à la navigation de l'utilisateur dans l'historique

```
window.addEventListener("popstate",
function(e) {
loadContent(location.pathname);
});
```

SUPPLEMENT : Fichiers et Drag & Drop

File Api

- L'Api File, va nous permettre de faire sélectionner à l'utilisateur des fichiers locaux , et d'accéder à leur contenu sans les transferer sur un serveur.

- 2 méthodes pour accéder aux fichiers :
  - Input de type file
  - Drag and drop

File Api

- L'api File nous permet d'accéder à un tableau de fichiers, qui représente les différents fichiers sélectionnés par l'utilisateur.

- Chaque fichier contenu dans ce tableau, possède plusieurs attributs :
  - Name : le nom du fichier
  - Type : le type mime du fichier
  - Size : la taille du fichier

File Api : file, fileList

- Exemple :

```
<input type="file" id="mesFichiers" multiple="true"/>

<script>
function fileAccess(e){
    var fichiers = e.target.files;
    for(var i = 0  ;  i < fichiers.length  ; i++){
        console.log(fichiers[i].name);
        console.log(fichiers[i].size);
        console.log(fichiers[i].type)
    }
}
var fileInput=document.querySelector("#mesFichiers");
fileInput.addEventListener("change",fileAccess);
</script>
```

File Api : file reader

- L’objet FileReader va permettre à notre navigateur de lire de manière asynchrone le contenu d’un fichier.
- Ce fichier peut provenir d’un input de type file ou être le fruit d’un drag and drop.

File Api : file reader

- L'objet FileReader possède un certain nombre de méthodes:
  - readAsDataURL(): lit le fichier et renvoie son contenu encodé en base 64
  - readAsText(): renvoie le contenu du fichier non encodé

File Api : file reader

- Les événements javascript liés a l'objet FileReader.
  - onloadstart
  - onload
  - onerror

File Api : file reader

- Exemple d'utilisation

```
<input type="file" id="mesFichiers" />

<script>
function fileAccess(e){
    var fichier = e.target.files[0];
    var reader = new FileReader();
    reader.onload = function(evt){
        console.log(evt.target.result)
    }
    reader.readAsDataURL(fichier);
}
var fileInput=document.querySelector("#mesFichiers");
fileInput.addEventListener("change",fileAccess);
</script>
```

Drag And Drop Api

- HTML5 nous propose une api qui va permettre de contourner une des plus ancienne interdiction du web : le fait de pouvoir « drag and dropper » des éléments d'une page web.

Drag And Drop Api

- L’attribut draggable mis a true permet de désactiver l’interdiction native du navigateur.
- Pour définir une zone de destination, il suffit d’ajouter un événement drop sur l’élément concerné.

Drag And Drop Api : Events

- Il existe plusieurs événements associés à cette fonctionalité :
  - **dragstart** : se déclenche lorsqu'un drag and drop d'une zone est commencé.
  - **dragover** : déclenché lorsqu'un drag and drop se situe au dessus de la zone concernée (potentiellement à répétion...).
  - **drop** : déclenché lorsqu'un élément en cours de déplacement est relaché au dessus de la zone concernée.

Drag And Drop Api : Events

- **dragend** : déclenché lorsqu'un drag and drop a été réalisé ou annulé.
- **drag** : déclenché pendant le déplacement (potentiellement à répétition...).
- **dragenter** : déclenché lorsqu'un drag and drop entre au dessus de la zone concernée.
- **dragleave** : déclenché lorsqu'un drag and drop quitte la zone concernée.

Drag And Drop Api : Exemple

```
<div id="element" draggable="true"></div>
<div class="box"></div>
<script>
var elem = document.querySelector("#element");
var box = document.querySelector(".box");
elem.addEventListener("dragstart",function(e){
e.dataTransfer.effectAllowed="move";
e.dataTransfer.setData("ElementId",e.target.id);
})
box.addEventListener("dragover",function(e){
    e.preventDefault();
})
box.addEventListener("drop",function(e){
           e.preventDefault();
          var elem = e.dataTransfer.getData("ElementId");
e.target.appendChild(document.getElementById(elem));})
</script>
```

Drag And Drop : File

- L'api nous propose également de permettre à nos utilisateurs de drag and dropper des fichiers de leur bureau jusque dans le navigateur.

- Pour ce faire, nous allons pouvoir récupérer un tableau de fichier dans le callback de l'événement drop.

- Le reste de la manipulation ne change pas par rapport à ce qu'on a vu précédemment.

Drag And Drop : File : Exemple

```
<script>
box.addEventListener("dragover",functio
n(e){
   e.preventDefault();
})
box.addEventListener("drop",function(e)
{
   e.preventDefault();
   var files = e.dataTransfer.files;
   for(var i = 0 ; i < files.length ; i+
```

SUPPLEMENT Dessiner en HTML

Canvas

- Dessine-moi un … canvas.

```
<canvas id="canvas" width="300" height="225"></canvas>
```

- Par défaut, un peu vide… remplissons le !

```
function draw() {
    var canvas = document.getElementById("canvas");
    var context = canvas.getContext("2d");
    context.fillRect(50, 25, 150, 100);
}
```

Canvas

- Nous pouvons donc manipuler notre canvas grâce à son contexte.
- Plusieurs choix sont à notre disposition /
  - la propriété fillStyle,
  - la propriété strokeStyle,
  - la méthode strokeRect(x,y,width,height),
  - la méthode fillRect(x,y,width,height),
  - la méthode clearRect(x,y,width,height).
- Peut-on effacer les propriétés d'un Canvas ?
  - en utilisant la méthode clearRect de la taille du canvas.

Canvas

- Le canvas est une grille en 2D
- La coordonnée (0,0) est le coin haut-gauche

Canvas

- Dessinons des Paths (lignes)

- Trois méthodes :
  - moveTo(x,y): se déplace vers
  - lineTo(x,y): dessine vers
  - Stroke() : trace les traits précédemment préparés

Canvas

- Ajoutons maintenant un peu de texte.

```
context.font = "bold 12px sans-serif";
context.fillText("hello", 200, 40);
context.fillText("world", 60, 160);
```

Canvas

- Les Gradients (fondu)

- Nous avons vu comment faire des lignes (path), des rectangles (simple shapes)
- Créons maintenant un fondu de gauche à droite
- Méthode de contexte: createLinearGradient(x1, y1, x2, y2).

```
var my_gradient = context.createLinearGradient(0, 0, 300, 0);
my_gradient.addColorStop(0, "black");
my_gradient.addColorStop(1, "white");

context.fillStyle = my_gradient;
context.fillRect(0, 0, 300, 225);
```

Images

- Le contexte a pour méthodes:
  - drawImage(image, dx, dy)
  - drawImage(image, dx, dy, dw, dh)
  - drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh): permet d'insérer une image dans un rectangle.

- Pour insérer une image en JS j'insére une image en html:

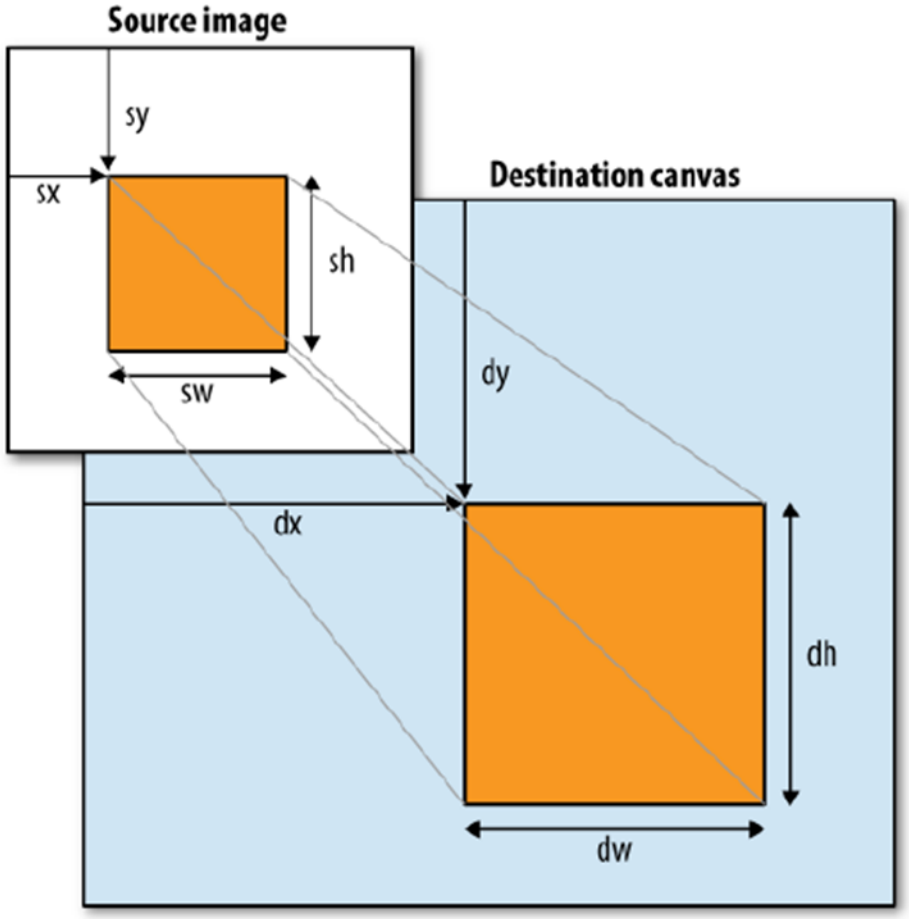
```
<img id="cat" src="image/cat.png" alt="Le chat">
```

- Puis j'insére le code javascript adapté

```
function draw() {
    var canvas = document.getElementById("canvas");
    var context = canvas.getContext("2d");
    var cat = document.getElementById("cat");

    context.drawImage(cat, 0, 0);
}
```

Images



Images

- Si nous créons notre image entièrement en JS, il nous suffit de faire :

```
<canvas id="e" width="177" height="113">
    Canvas non supporté par votre navigateur
</canvas>

<script>
var canvas = document.getElementById('e');
var context = canvas.getContext('2d');
var cat = new Image();
cat.src = 'images/cat.png';
cat.onload = function () {
    context.drawImage(cat, 0, 0);
};
</script>
```