Technologies WEB

Laurent Jouanneau
Course on Mozilla Education and Technologies @ Evry
Decembre, 2012

à propos de support

- attention support cross-browser
- caniuse.com
- developer.mozilla.org

installation serveur web

- cle usb
  - wamp / mamp

DOM Storage

DOM Storage

- Stockage de contenu dans le navigateur
  - sessionStorage : pendant la durée de vie d'une page dans le navigateur
  - localStorage : stockage persistant
  - globalStorage : obsolète
- contenu = clé/valeur
- API :
  - length
  - string key(long index); // Name of the key at index
  - string getItem(string key);
  - void setItem(string key, string data);
  - void removeItem(string key);
  - void clear();

DOM Storage

- stockage limité en volume (5 Mb) pour localStorage, pas de limite pour sessionStorage sur Firefox. Limites changeante en fonction du navigateur
- Accès limité aux pages d'un même domaine

indexedDB

indexedDB

- stockage gros volume, tout type de données : valeurs JS, blob, fichiers
- données = clé : valeur
- base de données transactionnelle
- API asynchrone
- Pas de SQL
- comme DOM Storage, accès limité au domaine de la page
- Attention : experimental
- Désactivé en mode privé

indexedDB

- Chaque base est identifié par un nom
- Une base contient un ou plusieurs « object store », contenant eux même des pairs de clé/valeur et des index
  - object store : similaire à une table en SQL
- Les valeurs : toute valeur JS, ou blob / fichier
- Les clés :
  - type : chaine, date, float ou array
  - créée ou pas à partir d'un key generator → générateur de séquence de valeurs
  - stockée ou pas dans la valeur, donc dans un objet. Il faut indiquer le « key path » = la propriété correspondante

indexedDB : request

- Pour chaque opération sur la base :
  - exécutée en asynchrone
  - un objet request est fourni
- request reçoit les résultats par évènements : il faut indiquer des listeners
  - myrequest.onerror = function(event) {…}
  - myrequest.onsuccess = function(event) {…}
- la propriété myrequest.result contient le résultat. sa nature dépend de l'opération demandée

indexedDB : transaction

- Toutes opérations est effectuée dans une transaction
  - si il y a une erreur non traitée : rollback
- définir des listeners avant de faire des opérations
  - mytransaction.oncomplete = ...
  - mytransaction.onerror = ...
- récupération des objects store à partir de la transaction
- opérations sur les objects store

indexedDB : utilisation générale

- Pour effectuer une opération
  - récupération de la base de donnée avec indexedDb
  - création d'une transaction « readonly » ou « readwrite »
  - récupération de l'object store via la transaction
  - lancement d'une opération sur l'object store : renvoi un objet request
  - déclaration des listeners sur request pour effectuer un traitement après l'opération

indexedDB : traitement des erreurs

- Les erreurs sont des événements qui se propagent sur les objets suivants
    1) objet request
    2) objet transaction
    3) objet database
- event.preventDefault() dans le request évite à la transaction de s’arrêter (pas de rollback)

indexedDB : ouverture / creation

- à l'ouverture de la base , une transaction « versionchange » est créé si la base n'existe pas ou si elle nécessite une mise à jour. Et un événement upgradeneeded est émis sur le request

```
var request = indexedDB.open('myAwesomeDatabase');
request.onupgradeneeded = function(event) {
    var db = event.target.transaction.db; // ou event.target.result

    var store = db.createObjectStore("voitures", {keyPath:'', autoIncrement:true});

    store.createIndex('modeles', 'modelRef', false);

    store.add({modelRef:'aa', name:'bb', dateSortie:'', puissance:'123'})

}
request.onsuccess = function(event) { ... }
request.onerror = function(event) { ... }
request.onblocked = function(event) { ... }
```

indexedDB : add/delete

```
var request = indexedDB.open('myAwesomeDatabase');
request.onupgradeneeded = function(event) {...}
request.onsuccess = function(event) {

     var db = event.target.result ;

    //création d'une transaction
    var transaction = db.transaction(['voitures'], "readwrite");

    transaction.oncomplete = function(event) {...}
    transaction.onerror = function(event) { … }

    // recupération du store
    var store = transaction.objectStore('voitures');

    var reqAdd = store.add({...})  // ou store.put({...})
    reqAdd.onsuccess = ...   // reqAdd.result = clé

    var reqDel = store.delete('cle')
    reqAdd.onsuccess = ...

}
request.onerror = function(event) { ... }
request.onblocked = function(event) { ... }
```

indexedDB : get

```
var request = indexedDB.open('myAwesomeDatabase');
request.onupgradeneeded = function(event) {...}
request.onsuccess = function(event) {

     var db = event.target.result ;

    //création d'une transaction
    var transaction = db.transaction(['voitures'], "readonly");

    transaction.oncomplete = function(event) {...}
    transaction.onerror = function(event) { … }

    // recupération du store
    var store = transaction.objectStore('voitures');

   var request = store.get('cle');
   request.onsuccess = ...  // request.result = valeur
   request.onerror = ...

}
request.onerror = function(event) { ... }
request.onblocked = function(event) { ... }
```

indexedDB : cursor

```
var request = indexedDB.open('myAwesomeDatabase');

request.onsuccess = function(event) {

     var db = event.target.result ;

    //création d'une transaction
    var transaction = db.transaction(['voitures'], "readonly");

    transaction.oncomplete = function(event) {...}
    transaction.onerror = function(event) { … }

    // recupération du store
    var store = transaction.objectStore('voitures');
    store.openCursor().onsuccess = function(event) {
      var cursor = event.target.result;
      if (cursor) {
        vehicles.push(cursor.value);
        cursor.continue(); // indique de rappeler onsuccess
      }
    };
}
```

indexedDB : cursor sur index

```
    var store = transaction.objectStore('voitures');

    var index = store.index('modeles') ;

    index.openCursor().onsuccess = function(event) {
      var cursor = event.target.result;
      if (cursor) {
        vehicles.push(cursor.value);
        cursor.continue(); // indique de rappeler onsuccess
      }
    };
```

```
// avec un range
    var boundKeyRange = IDBKeyRange.bound("Peugeot", "Renault", false, false);
    var store = transaction.objectStore('voitures');
    var index = store.index('constructeurs') ;

    index.openCursor(boundKeyRange).onsuccess = function(event) {
      var cursor = event.target.result;
      if (cursor) {
        vehicles.push(cursor.value);
        cursor.continue();
      }
    };
```

XmlHttpRequest

xmlHttpRequest

- permet de faire des requêtes HTTP
- requêtes synchrones fortement déconseillées
- initialisation
  - créer un objet xmlHttpRequest
  - définir les callback pour recevoir les résultats
- Lancement de la requête
  - ouvrir la connexion avec open()
  - envoyer le contenu avec send()
- récupération des résultats via les propriétés response, responseText ou responseXml

xmlHttpRequest : init

- méthode traditionnelle : listener sur readystatechanged

```
var req = new XMLHttpRequest();
req.onreadystatechange = function () {
    if (req.readyState == 4) {
        if(req.status == 200)
          alert(req.responseText);
        else
          alert("Error loading content\n");
    }
};
```

- readyState : 0=rien, 1=open, 2=headers, 3=loading, 4=loaded

xmlHttpRequest : init listeners

- Méthode « moderne » : listeners sur les évènements loadstart, progress, error, abort, load, loadend

```
function onProgress(evt) {
    if (evt.lengthComputable) {
        var percentComplete = evt.loaded / evt.total;
        // ...
    }
}
function onLoad(evt)  { alert("The transfer is complete.");}
function onError(evt) { alert("An error occurred while transferring the file.");}
function onAbort(evt) { alert("The transfer has been canceled by the user.");}

req.addEventListener("progress", onProgress, false);
req.addEventListener("load", onLoad, false);
req.addEventListener("error", onError, false);
req.addEventListener("abort", onAbort, false);
```

xmlHttpRequest : request

- méthod GET

```
req.open('GET', 'http://xulfr.org/', true);
req.send(null);
```

```
void send();
void send(DOMString data);
void send(ArrayBuffer data);
void send(Blob data);
void send(Document data);
void send(FormData data);
```

- méthode POST

```
req.open('POST', 'http://monsite.tld/mon_script.php', true);
req.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
req.send("param1=val1&param2=val2");
```

- avec FormData

```
var docform = document.getElementById('formulaire')
var oMyForm = new FormData(docform);
req.send(oMyForm);
```

```
var oMyForm = new FormData();
oMyForm.append("param1", "va1");
oMyForm.append("param2", 1234);
var b = new Blob("filecontent", {type :"image/jpeg"})
oMyForm.append("param3", b);
req.send(oMyForm);
```

xmlHttpRequest : response

- req.responseType : indique le type de données attendues. « arraybuffer », « blob », « document », « json », « text », « moz-blob »
- req.response
- req.responseText (texte ou html)
- req.responseXml

HTTP access control

- permet d'accéder à une ressource fournie par un autre domaine que celui de la page
- CSS, images, scripts, chargés par la page : autorisé par défaut
- XHR, web fonts, webgl textures, images utilisées dans <canvas> :
  - le navigateur demande une autorisation. Rien à faire coté page web, sauf gérer les cas d'erreur
  - le serveur doit renvoyer des en-têtes spécifiques autorisant ou non la récupération du contenu
  - plusieurs scénarios possibles selon la nature de la requête : simple requête, requête avec contrôle en amont, requête avec identification

HTTP Access Control

- méthode GET, POST avec contenu application/x-www-form-urlencoded, multipart/form-data, ou text/plain
  - → simple request
- méthode PUT, DELETE ou POST avec autre contenu, ou en-têtes personnalisés
  - → preflight request
- envoi de cookies ou authentification HTTP, en simple request ou preflight
  - → credentials request

HTTP Access Control : simple request

- En-tête requête :
  - Origin : http://site-appelant/
- En-têtes possibles dans la réponse du serveur

```
# indique si tout site (*) ou seulement le site <origin> a le droit d'utiliser
# le contenu demandé. si header absent ou origin différente : autorisation
# refusée

Access-Control-Allow-Origin: <origin> | *

# indique les en-têtes personnalisés qui peuvent être lu par XHR

Access-Control-Expose-Headers: X-My-Custom-Header, X-Another-Custom-Header
```

HTTP Access Control : preflight request

- une première requête avec la méthode OPTIONS pour demander l'autorisation
- En-tête requête :

```
Origin : http://site-appelant/
# la méthode demandée
Access-Control-Request-Method: <method>
# la liste des en-têtes personnalisés à envoyer
Access-Control-Request-Headers: <field-name>[, <field-name>]*
```

- En-têtes possibles dans la réponse du serveur

```
Access-Control-Allow-Origin: <origin> | *
Access-Control-Expose-Headers: X-My-Custom-Header, X-Another-Custom-Header

# indique la durée que le nav peut mettre en cache ces infos
Access-Control-Max-Age: <delta-seconds>

# indique les méthodes autorisées
Access-Control-Allow-Methods: <method>[, <method>]*

# indique les en-têtes autorisés
Access-Control-Allow-Headers: <field-name>[, <field-name>]*
```

HTTP Access Control : preflight request

- la 1ère réponse ne renvoi pas de contenu
- la 1ère requête/réponse est transparente pour le développeur
- Si l'autorisation est ok, Une 2ième requête (la « vraie ») est envoyée, comme une simple request.
- XHR reçoit alors cette réponse.

HTTP Access Control : credentials

- Pour indiquer que l'on veut aussi envoyer les cookies et/ou authentification HTTP
  - xhr.withCredentials = true;
- Le browser envoi la « simple request » ou « preflight request » avec les cookies et/ou info auth
- Pour que le contenu soit accessible à la page web, le serveur doit répondre par
  - Access-Control-Allow-Credentials: true

File API

File API

- Permet de lire le contenu d'un fichier et de le manipuler **dans** la page web
- <input type="file" id="selector" onchange=""/>
  - propriété files = objet FileList (~ array)
  - -> [file, file, file] file = objet File
- propriétés objet File : name, size, lastModifiedDate, type

```
Obsolète :
fileName
fileSize
getAsBinary()
getAsDataUrl()
getAsText()
```

- Utilisation de FileReader pour lire le contenu

File API : FileReader

- var reader = new FileReader();
- Lecture d'un fichier

```
void abort();
void readAsArrayBuffer(in Blob blob);
void readAsBinaryString(in Blob blob);
void readAsDataURL(in Blob file);
void readAsText(in Blob blob, [optional] in DOMString encoding);
```

- Event « load » quand la lecture est fini → listener
- propriété result = contenu
- le contenu peut être injecté dans la propriété src d'une image par exemple

Drag'n'drop

Drag'n'drop

- Les éléments que l'on peut « glisser-déposer » : draggable="true"
- le glissage d'un élément démarre, il reçoit les événements **dragstart, drag, dragend**
- Événements sur les éléments survolés pendant le glissage : dragenter, dragover, dragleave
- À la dépose, événement drop sur l'élément survolé
- Sur les objets event, propriété dataTransfer contenant les données à « transférer », type de DND etc.

Drag'n'drop

- sur dragstart, on va initialiser :
  - event.dataTransfer.effectAllowed
  - event.dataTransfer.setData(...)
  - event.dataTransfer.setDragImage(...)
- Sur dragenter/dragover : vérifier le type de glisser-deposer et faire un event.preventDefault() si le drop est autorisé
- sur le drop : event.dataTransfer.getData() event.preventDefault() pour annuler l'action drop par défaut du navigateur
- sur le dragend : event.dataTransfer.dropEffect vaut « none » si DND annulé

Drag'n'drop

- Un glissage peut venir de « l'exterieur » : il n'y a pas à traiter dragstart, drag, dragend
- dataTransfer.files : liste de fichier que l'utilisateur glisse/depose sur l’élément de la page web

Media queries

Media Queries

- permet d'appliquer des styles en fonction des caractéristiques de l'appareil
  - ex : adaptation de l'affichage du contenu en fonction de la taille de l'écran

```
@media  all and (max-width : 500px) {

    /* regles css à appliquer quand la fenêtre
     a moins de 500px, pour tout type de device */

}
```

- all = media type
- (max-width:500px) = expression sur des media features
- and = operateur. « not » est aussi possible

Media Queries

- media type : all, aural, braille, handeld, print, projection, screen, tty, tv, embossed
- media features : width, height, device-width, device-height, aspect-ratio, device-aspect-ratio, color, color-index, resolution...
  - idem avec min- et max-

```
<link rel="stylesheet" media="screen and (max-device-width: 799px)" />

@media handeld and (orientation: portrait) and (device-aspect-ratio: 16/9),
    not projection and  (aspect-ratio: 4/3),
    screen and (device-aspect-ratio: 16/9)
    { ... }
```

<audio>

- balise pour jouer du son
- quelques attributs
  - src (fichier wav ou ogg)
  - autoplay
  - loop
- api
  - play(), pause()

WebWorker

Web Worker

- Worker ~= thread
  - Permet de lancer des scripts en parallèle
- Le code d'un worker
  - ne peut pas manipuler du DOM
  - est dans un fichier dédié
- Les échanges de données entre la page et le worker se font via des messages

Web worker

- Dans la page web

```
// creation du worker
// mycode.js n'est pas encore executé

var worker = new Worker("mycode.js");

// listener pour les messages en provenance de mycode.js
worker.onmessage = function(event) {
    var msg = event.data;

};

worker.onerror = function(event) {
  var error =  event.data
  throw event.data;
};

// premier message = lancement du worker
worker.postMessage("");


// pour arrete le worker à un moment donné
worker.terminate() ;
```

Web worker

- Dans le code du worker

```
onmessage = function(event) {
  var msg = event.data; // si on a besoin de paramètres

  if (msg == "") {
      var result = lancementCalcul() ;
      postMessage({resultat : result,
                msg :"un objet comme message qui sera transmis en"});
  }
  else if (msg == "foo") {
   …
   }
}

/* fonctions disponibles
atob() btoa()
setInterval() clearInterval()
setTimeout() clearTimeout()
dump()
XMLHttpRequest()
Worker()
importScripts()
FileReaderSync()

*/
```