JavaScript
BOM*

IFNTI Sokodé – L3

26 Avril 2021

Il y a des API que les navigateurs mettent à disposition. Nous allons, entre autre, nous concentrer sur cell permettant de manipuler le Browser (le navigateur) et également le document HTML.

1 API ?

Vous avez déjà vu ce qu'est une API. Il n'y aura ici que des rappels. Le principe d'une API est de mettre à disposition des controls simplifiés d'opérations complexes réalisé par l'application (dans le cas du WEB, le navigateur mais cela peut-être des services hébergés sur des serveurs).

Quelques exemples :
- les API intégrées aux navigateurs peuvent donner accès :
  - au document html (DOM)
  - à des données de géolocalisation
  - à des données graphiques
- les API externes peuvent êtres :
  - l'API de google maps pour afficher une carte interactives
  - l'API de tweeter pour afficher certains tweet
  - l'API de youtube pour intégrer une vidéo

Nous nous concentrerons ici sur certaines API proposées par les navigateurs. Ces API tournent généralement autour de la définition d'objet et sa manipulation avec des méthodes associées.

2 BOM

Le **B**rowser **O**bject **M**odel est une API regroupant plusieurs autres API qui elles mêmes peuvent contenir d'autres API ...

Le BOM expose la fenêtre du navigateur à travers l'interface `Window`. L'objet `Window` implémente l'interface du même nom. Cet objet est supporté par tous les navigateurs. Toutes les entités globales sont contenues dans `Window` En quelque sorte c'est la racine de l'API du BOM.

Chaque onglet possède son propre objet `Window`. Il n'est généralement pas nécessaire de le mentionner pour utiliser ses éléments. C'est un objet dit « implicte ».

Une petite liste d'objet appartenant au BOM et étant donc tous inclus dans `Window`.
- L'objet `Navigator` qui représente le navigateur
- L'objet `History` qui représente l'historique de navigation
- L'objet `Location` qui représente l'URL de la page courante
- L'objet `Screen` qui permet de connaître quelque propriété de l'écran qui affiche la page
- L'objet `Document` qui nous permet de manipuler le DOM.

3 L'objet Window

Comme mentionné précédemment `Window` représente la fenêtre du navigateur actuellement ouverte.

*Fortement inspiré de `https://www.pierre-giraud.com/javascript-apprendre-coder-cours/browser-object-model-window/`

3.1 Propriétés

Nous n'allons pas énumérer l'ensemble des propriétés de l'objet `Window`. Nous en avons déjà cité quelques uns comme les propriétés `document`, `navigator`, `location` .... Concentrons nous sur 4 :

— innerHeight
— innerWidth
— outerHeight
— outerWidth

Les propriétés « outer » concerne les dimensions du navigateur en prenant en compte les options, boutons et autres layout périphériques du navigateur.

Les propriétés « inner » concerne les dimensions de la partie rendu de la fenêtre (là ou l'on va afficher notre page).

Exercice Faite une page qui affiche les dimensions du navigateur.

Question Mais à quoi cela peut-il nous servir ?

3.2 Méthodes

Évidemment, il y a un grand nombre de méthodes dans l'objet `Window` et nous n'allons pas tout couvrir. Concentrons nous sur :

— ouvrir une fenêtre
— fermer une fenêtre
— redimensionner une fenêtre
— déplacer une fenêtre

3.2.1 open et close

La méthode `open()` permet d'ouvrir une ressource. La méthode prend l'url de la ressource à ouvrir en argument. `open()` va aussi prendre en paramètre l'identifiant de la fenêtre dans laquelle ouvrir la ressource. Si aucun nom de convient, une nouvelle fenêtre sera créée.

Le dernier argument est une chaîne de caractère pouvant contenir une liste de paramètre pour spécifier en finesse les caractéristiques de la fenêtre. :

— la position de la fenêtre
— la taille (avec width et height)
— si elle peut être redimensionnée.

`open()` retourne une référence pointant vers la fenêtre créée.

`close()` ferme la fenêtre.

Exercice Hey et si on faisait une page qui présente un bouton. Quand le bouton est cliqué, cela ouvre la page de l'exercice précédent avec une résolution de 500x500.

3.2.2 resize et move

Pour redimensionner la fenêtre que nous avons, nous pouvons utiliser les méthodes : `resizeBy()` et `resizeTo()` qui s'appliquent sur la référence retournée par `open()`.

Exercice ... On essaie ? Créons deux boutons de redimensionnement de cette belle fenêtre.

Bon cela ne marche pas ... pourquoi ? Comment pourrions nous remedier au problème ? Pour l'instant, il est possible de « corriger » le problème en retirant l'url de la méthode open. La page ne contiendra plus rien ... mais pour notre exercice, cela est peu important.

move a votre avis que font les méthode `moveBy` et `moveTo`... Ajoutez les boutons correspondant.

scroll de la même manière, il y a `scrollBy` et `scrollTo`. Mais il faut un peut de contenu pour que l'on puisse effectivement faire défiler la page.

3.2.3 Quelques popup

Il y a d'autres méthode faisant apparaître des fenêtre :
— `alert()` qui fait apparaitre un message
— `prompt()` qui demande une saisie et la retourne
— `confirm()` qui demande une confirmation et renvoie un booléen.

4 L'objet Navigator

L'objet `navigator` contient les informations sur le navigateur en cours d'utilisation. Il est aussi appelé l'agent utilisateur (« user agent »).

Attention, ici nous manipulons directement des données de l'utilisateur. Il y a maintenant beaucoup de loi qui encadre l'accès à ces données qui doivent généralement être soumises au consentement de l'utilisateur.

Nous trouverons :
— language : la langue définie dans le navigateur
— geolocation : la position de l'utilisateur
— cookieEnabled : si les cookies sont autorisés ou non
— platform : la plateforme utilisée par l'utilisateur
— ... bien d'autres information (nom du navigateur, version,...)
Attention, c'est le navigateur qui met à disposition ces informations, il peut très bien :
— se tromper
— retourner de mauvaises informations intentionnellement (cela s'est fait car certains site refusait certains navigateurs)

Exercice Et si on faisait une petite page pour tester ça ?

5 Geolocation

L'interface `geolocation` permet d'avoir (sous réserve de l'acceptation de l'utilisateur) la position de l'appareil. L'objet `geolocation` est un attribut de l'objet `navigator`. Pour des raisons de sécurité ces méthodes ne sont accessibles que dans un context sécurisé (HTTPS).
— `getCurrentPosition()` prend en paramètre une fonction qui est appelé avec comme paramètre la position de l'appareil sous la forme d'un objet `Position` qui est .
— `watchPosition()` permet de définir une fonction qui sera appelée lorsque l'appareil bouge. Cette méthode retourn un `id`.
— `clearWatch()` utilise cet `id` pour supprimer la fonction passée à `watchPosition()`.
La méthode `getCurrentPosition()` retourne un objet `position` ce dernier implémente deux propriétés :
— `coords` qui contient les coordonnées de la position lui même possédant les propriétés :
  — latitude
  — longitude
  — altitude
  — ...
— `timestamp` qui représente le moment où la position a été acquise.

Exercice Où sommes nous ? Et si on faisait une page ?

6 History

L'objet `history` permet de manipuler l'historique de l'onglet en cours. Grâce à lui, il est possible de revenir à la page précédente sans avoir besoin d'explicitement s'en rappeler.

Nous trouvons comme méthodes et propriétés :
— `length` retourne le nombre d'URL dans l'historique en comptant l'url actuelle
— `go()` charge une page de l'historique. Cette page est spécifiée par son rang dans l'historique par rapport à la page actuelle (ex : -1 précédente, +1 suivante)
— `back()` chargement de la page précédente. A quoi-est-ce équivalent ?
— `forward()` chargement de la page suivante... Même question.

7 Location

L'objet `Location` permet d'accéder à des informations liées à l'URL de la page. L'objet `location` peut être accéder soit à partir de l'interface `window` ou `Document`

Les propriétés sont :
— `protocole` retourne le protocole de l'url
— `hostname` retourne le nom de l'hôte de l'url
— `port` retourne le port de l'url
— certaines combinaisons sont possibles :
  — `host` retourne le nom de l'hôte ainsi que le port
  — `origin` retourne le nom de l'hôtele port et le protocol
— `pathname` retourne le chemin
— `search` retourne la partie contenant les arguments (e.g. ?param1=value1)
— `hash` retourne l'ancre d'une url
Trois méthodes importantes de `Location` sont :
— `reload()` va recharger la page
— `assign()` va charger une ressource à partir de l'URL passée en argument.
— `replace()` va remplacer le document actuel par un autre (URL fournie en argument). Différence avec `assign()` est que `replace()` n'enregistre pas la page dans l'historique.

Exercice Et si nous expérimentions la différence nous même ?