

Formation Programmation Multi- Plateforme

TP - Service Worker

Table of Contents

Présentation.....	1
Cycle de vie.....	1
Enregistrement.....	1
Installation.....	2
Activation.....	2
Service Worker par l'exemple.....	2
Installation.....	2
Page par défaut.....	3
Enregistrer un Service Worker.....	4
Cycle de vie d'un Service Worker.....	4
Intercepter les requêtes.....	5
Mise en cache des fichiers.....	5
Utiliser le cache.....	6
Mettre une réponse dans le cache.....	8
Faire évoluer le cache.....	10
Mode hors ligne.....	10
Données métiers.....	11
Communication via des messages.....	12

Présentation

Un *Service Worker* est une catégorie de *Web Worker*.

Il s'agit essentiellement d'un fichier JavaScript qui :

- s'exécute de manière séparée du thread principal du navigateur.
- est capable d'intercepter les requêtes allant vers le réseau.
- permet de mettre en cache des ressources et de les récupérer du cache.

Quelques caractéristiques :

- Un *Service Worker* est conçu pour être fondamentalement asynchrone. Les API bloquantes comme une requête *XHR synchrone* ou le *localStorage* ne peut être utilisé.
- Un *Service Worker* peut recevoir des messages depuis un serveur même si l'application n'est pas active.
- Un *Service Worker* ne peut accéder directement au DOM. La communication avec la page s'effectue via des messages.
- Un *Service Worker* fonctionne uniquement sur HTTPS.

Cycle de vie

Trois étapes du cycle de vie :

- Enregistrement
- Installation
- Activation

Enregistrement

```
if ('serviceWorker' in navigator) {  
  
    // enregistrement d'un service worker  
    navigator.serviceWorker.register('/service-worker.js')  
  
    .then(function(registration) {  
        console.log('Enregistrement Ok, le scope est :', registration.scope);  
    })  
  
    .catch(function(error) {  
        console.log('Enregistrement Ko, erreur:', error);  
    });  
}
```

Lors de l'enregistrement, il est possible de spécifier un *scope* différent :

```
// Les requêtes /off-app/, /off-app/res, ... seront contrôlées par le service worker
navigator.serviceWorker.register('/service-worker.js', {
  scope: '/off-app/'
})
```

Installation

L'installation a lieu après l'enregistrement et si le *service worker* est considéré comme nouveau par le navigateur.

Un événement *install* est déclenché dans cette phase.

```
// Traitement de l'événement 'install' du service worker
self.addEventListener('install', function(event) {
  // mise en cache des ressources
});
```

Activation

Un *service worker* va osciller entre une phase d'activation et une phase d'attente.

```
// Traitement de l'événement 'activate' du service worker
self.addEventListener('activate', function(event) {
  // ...
});
```

Service Worker par l'exemple

Installation

- Créer un répertoire *02-service-worker* :

```
mkdir 02-service-worker
```

- Initialiser un projet *npm* :

```
npm init -y
```

- Installer un serveur :

```
npm i http-server -D
```

- Compléter le fichier *package.json* pour ajouter la tâche *start* :

package.json

```
{
  ...
  "scripts": {
    "start" : "http-server -a localhost -c 0",
    ...
  },
  ...
}
```

- Démarrer le serveur :

```
npm start
```

Page par défaut

- Ajouter une page Web *index.html* :

index.html

```
<!doctype html>
<html>
  <head>
    <link rel="stylesheet" href="app.css">
  </head>
  <body>
    <h1>Service Worker</h1>

    <script src="app.js"></script>
  </body>
</html>
```

- Ajouter les fichiers *app.js* et *app.css*. A ce stade, l'arborescence est la suivante :

```
/02-service-worker
  index.html
  app.js
  app.css
  package.json
  /node_modules/
```

Enregistrer un Service Worker

- Compléter le fichier *app.js* comme suit :

app.js

```
(function () {  
  'use strict';  
  
  if (!('serviceWorker' in navigator)) {  
    console.log('Service worker non supporté');  
    return;  
  }  
  navigator.serviceWorker.register('sw.js')  
    .then(() => {  
      console.log('Enregistrement OK');  
    })  
    .catch(error => {  
      console.log('Enregistrement KO :', error);  
    });  
})();
```

- Créer un fichier *sw.js*.
- Afficher la page d'accueil. Visualiser la console du navigateur :

Enregistrement OK

Cycle de vie d'un Service Worker

- Compléter le fichier *sw.js* :

sw.js

```
self.addEventListener('install', event => {  
  console.log('Installation du Service Worker...');  
});  
  
self.addEventListener('activate', event => {  
  console.log('Activation du Service Worker...');  
});
```

- Recharger la page :

Enregistrement OK
Installation du Service Worker...

Par défaut le service worker n'est pas actif, il est en attente.

- Compléter le fichier `sw.js` :

`sw.js`

```
...  
  
// active immédiatement un nouveau service worker  
self.skipWaiting();
```

- Recharger la page :

```
Enregistrement OK  
Installation du Service Worker...  
Activation du Service Worker...
```

Intercepter les requêtes

- Compléter le fichier `sw.js` :

`sw.js`

```
...  
  
self.addEventListener('fetch', event => {  
  console.log('Fetching:', event.request.url);  
});
```

- Recharger plusieurs fois. Vérifier que les requêtes sont bien interceptées. :

```
Fetching: http://localhost:8080/  
Fetching: http://localhost:8080/app.css  
Fetching: http://localhost:8080/app.js  
Enregistrement OK
```

Pour en savoir plus :

- Événement Fetch Event : <https://developer.mozilla.org/en-US/docs/Web/API/FetchEvent>

Mise en cache des fichiers

La mise en cache des fichiers dans un *Service Worker* se fait l'aide de *Cache API* (<https://developer.mozilla.org/en-US/docs/Web/API/Cache>).

- Modifier le service worker comme-suit (modification du traitement lors de l'événement *install*) :

```
const FILES_TO_CACHE = [
  '.',
  'app.css',
  'app.js'
];

const STATIC_CACHE_NAME = 'pages-cache-v1';

self.addEventListener('install', event => {
  console.log('Installation du Service Worker...');
  console.log('Mise en cache des ressources');
  event.waitUntil(
    caches.open(STATIC_CACHE_NAME)
      .then(cache => {
        return cache.addAll(FILES_TO_CACHE);
      })
  );
});
```

- Recharger la page et vérifier le cache :

#	Request	Response
0	http://localhost:8080/	OK
1	http://localhost:8080/app.css	OK
2	http://localhost:8080/app.js	OK

Utiliser le cache

- Créer un fichier *app.no.cache.js*.
- Compléter le fichier *index.html* comme suit :


```
<!doctype html>
<html>
  <head>
    <link rel="stylesheet" href="app.css">
  </head>
  <body>
    <h1>Service Worker</h1>

    <script src="app.js"></script>
    <script src="app.no.cache.js"></script> ①
  </body>
</html>
```

① : inclusion du script *app.no.cache.js*.

- Modifier le service worker comme-suit (modification du traitement lors de l'événement *fetch*) :

```
self.addEventListener('fetch', event => {
  console.log('Fetching:', event.request.url);

  event.respondWith(
    caches.match(event.request).then(response => {
      if (response) {
        console.log(event.request.url, 'servi depuis le cache');
        return response;
      }
      console.log(event.request.url, 'servi depuis le réseau');
      return fetch(event.request)

    }).catch(error => {
      console.log("oops");
    })
  );
});
```

- Recharger la page plusieurs fois :

```
...
http://localhost:8080/ servi depuis le cache
http://localhost:8080/app.css servi depuis le cache
http://localhost:8080/app.js servi depuis le cache
http://localhost:8080/app.no.cache.js servi depuis le réseau
...
```

- Modifier le titre de la page *index.html* :

index.html

```
<!doctype html>
<html>
  <head>
    <link rel="stylesheet" href="app.css">
  </head>
  <body>
    <h1>Service Worker en cache</h1> ①

    <script src="app.js"></script>
    <script src="app.no.cache.js"></script>
  </body>
</html>
```

① : modifier le texte.

- Recharger la page. La modification n'est pas prise en compte.

Mettre une réponse dans le cache

- Créer un fichier *app.1.js*.
- Compléter le fichier *index.html* comme suit :

index.html

```
<!doctype html>
<html>
  <head>
    <link rel="stylesheet" href="app.css">
  </head>
  <body>
    <h1>Service Worker</h1>

    <script src="app.js"></script>
    <script src="app.no.cache.js"></script>
    <script src="app.1.js"></script> ①
  </body>
</html>
```

① : inclusion du script *app.1.js*.

- Modifier le service worker comme-suit (modification du traitement lors de l'événement *fetch*) :

```

self.addEventListener('fetch', event => {
  console.log('Fetching:', event.request.url);

  event.respondWith(
    caches.match(event.request).then(response => {
      if (response) {
        console.log(event.request.url, 'servi depuis le cache');
        return response;
      }
      console.log(event.request.url, 'servi depuis le réseau');
      return fetch(event.request)
    })

    // rubrique à ajouter
    .then(function (response) {

      return caches.open(STATIC_CACHE_NAME).then(cache => {

        // mise en cache des ressources qui ne contiennent pas no.cache
        if (event.request.url.indexOf('no.cache') < 0) {
          cache.put(event.request.url, response.clone());
        }
        return response;
      });
    })

    .catch(error => {
      console.log("oops");
    })
  );
});

```

- Au premier rechargement :

```

...
http://localhost:8080/app.no.cache.js servi depuis le réseau
...
http://localhost:8080/app.1.js servi depuis le réseau
...

```

- Lors des rechargements suivants :

```
...  
http://localhost:8080/app.no.cache.js servi depuis le réseau  
...  
http://localhost:8080/app.1.js servi depuis le cache  
...
```

Faire évoluer le cache

- Modifier le service worker comme-suit (modification du traitement lors de l'événement *activate*):

sw.js

```
self.addEventListener('activate', event => {  
  console.log('Activating new service worker...');  
  
  const cacheWhitelist = [STATIC_CACHE_NAME];  
  
  // suppression des caches excepté le cache courant (STATIC_CACHE_NAME)  
  event.waitUntil(  
    caches.keys().then(cacheNames => {  
      return Promise.all(  
        cacheNames.map(cacheName => {  
          if (cacheWhitelist.indexOf(cacheName) < 0) {  
            return caches.delete(cacheName);  
          }  
        })  
      );  
    })  
  );  
});
```

- Modifier le nom du cache :

sw.js

```
const STATIC_CACHE_NAME = 'pages-cache-v2';
```

- Recharger la page et vérifier que le cache *pages-cache-v1* est bien supprimé.

Mode hors ligne

- Arrêter le serveur.
- Fermer le navigateur.
- Démarrer le navigateur et relancer l'application <http://localhost:8080/>.
- Vérifier que la page s'affiche toujours.

Données métiers

Stockage des données métiers au chargement de l'application

Pour faciliter le stockage des données métiers, nous allons utiliser la librairie *localforage* (<https://localforage.github.io/localForage>).

- Modifier le service worker comme-suit :

sw.js

```
// import du script _localforage_

importScripts("https://cdn.rawgit.com/mozilla/localForage/master/dist/localforage.js")
;

....

// événement _install_

self.addEventListener('install', event => {
  console.log('Installation du Service Worker...');
  console.log('Mise en cache des ressources');
  event.waitUntil(
    Promise.all([
      caches.open(STATIC_CACHE_NAME)
        .then(cache => {
          return cache.addAll(FILE_TO_CACHE);
        }),
      fetch('https://raw.githubusercontent.com/DevInstitut/conference-
data/master/speakers.json')
        .then(resp => resp.json())
        .then(speakers => {
          localforage.config({storeName: 'speakers'})
          for (key in speakers) {
            localforage.setItem(key, speakers[key])
          }
        })
    ])
  );
});
```

- Recharger la page et vérifier le contenu de la base indexedDb.

Application		
Manifest		
Service Workers		
Clear storage		
Storage		
Local Storage		
Session Storage		
IndexedDB		
localforage - http://localhost:8080		
keyvaluepairs		
localforage-detect-blob-support		
speakers		
Web SQL		
Cookies		

#	Key	Value
0	*101*	▼ Object ► badges: [(-)] bio: "Yufeng is a Developer Advocate for the Google Cloud Platform" company: "Google" companyLogo: "/images/logos/google.svg" country: "USA" id: 101 name: "Yufeng Guo" photoUrl: "/images/speakers/Yufeng_Guo.png" shortBio: "Yufeng is a Developer Advocate for the Google Cloud Pla" ► socials: [(-)] ► tags: (2) ["Tensor Flow", "Cloud"]
1	*102*	► Object
2	*1028*	► Object
3	*103*	► Object

Communication via des messages

Une page et un *Worker* peuvent communiquer par le biais de message.

Côté page :

app.js

```
// recevoir des messages du service worker
navigator.serviceWorker.onmessage = function(event) {
  console.log("Reçu du SW : ", event.data);
}

// envoyer un message au service worker
if (navigator.serviceWorker.controller) {

  navigator.serviceWorker.controller.postMessage({
    "command": "MISE_A_JOUR",
    "message": "Hello je suis un client"
  });
}
```

Côté service worker :

app.js

```
// ecoute de message provenant d'un client

self.addEventListener('message', event => {
  // traitement du message (event.data)
})

// exemple d'envoi de message à tous les clients (en local)

self.clients.matchAll().then(function(clients) {
  clients.forEach(function(client) {
    client.postMessage({
      "command": "HELLO_LES_CLIENTS",
      "message": "Hello je suis un SW"
    });
  })
})
```