API Service Worker

Note : Cette fonctionnalité est disponible via les Web Workers.

Les *service workers* agissent principalement comme des serveurs intermédiaires entre les applications web, le navigateur, et le réseau (lorsque celui-ci est disponible). Ils sont conçus, entre autres, pour permettre la création de fonctionnalités hors ligne, intercepter les requêtes réseau, et agir en conséquence selon que le réseau est disponible ou non, et mettre à jour les fichiers qui sont situés sur le serveur. Ils permettent également d'accéder aux API de notifications *push* et de synchronisation en arrière-plan.

Note : Les *service workers* sont un type de *web worker*. Voir *Web workers* pour des informations générales sur les types de workers et leurs cas d'utilisation.

Concepts et utilisation des service workers

Un *service worker* est un *worker* piloté par les évènements, enregistré par rapport à une origine et un chemin. Il prend la forme d'un fichier JavaScript qui peut contrôler la page web/le site auquel il est associé, interceptant et modifiant les requêtes de navigation et de ressources, et mettant en cache les ressources de manière très granulaire pour vous donner un contrôle complet sur le comportement de votre application dans certaines situations (la plus évidente étant lorsque le réseau n'est pas disponible).

Un *service worker* s'exécute dans le contexte d'un *worker* et n'a donc pas accès au DOM. Il s'exécute dans un *thread* différent du *thread* JavaScript principal et n'est donc pas bloquant. Il est conçu pour fonctionner de façon complètement asynchrone. Aussi, les API synchrones comme XHR et *Web Storage* ne peuvent pas être utilisées dans le code d'un *service worker*.

Les *service workers* ne peuvent pas importer des modules JavaScript de manière dynamique, et `import()` générera une erreur si elle est appelée dans le contexte global d'un *service worker*. Les importations statiques utilisant l'instruction `import` sont autorisées.

Les *service workers* ne sont disponibles que dans des contextes sécurisés : cela signifie que leur document est servi avec HTTPS, bien que les navigateurs considèrent également `http://localhost` comme un contexte sécurisé, pour faciliter le développement local. Les connexions HTTP sont susceptibles d'être victimes d'injection de code malveillant par des attaques de l'homme du milieu, et de telles attaques pourraient être aggravées si elles avaient accès à ces API puissantes.

> **Note :** Sous Firefox, pour effectuer des tests, vous pouvez exécuter les *service workers* sur HTTP (de manière non sécurisée) ; il suffit de cocher l'option **Activer les Service Workers sur HTTP (lorsque la boîte à outils est ouverte)** dans le menu des options/roue dentée des outils de développement de Firefox.

> **Note :** Contrairement aux tentatives précédentes dans ce domaine telles que AppCache (angl.), les *service workers* ne font pas d'hypothèses sur ce que vous essayez de faire, mais échouent ensuite lorsque ces hypothèses ne sont pas exactement correctes. Au lieu de cela, les *service workers* vous donnent un contrôle bien plus granulaire.

> **Note :** Les *service workers* utilisent massivement les promesses, car en général ils attendent que des réponses arrivent, après quoi ils réagissent avec une action de succès ou d'échec. L'architecture des promesses est idéale pour cela.

Enregistrement

On commence par enregistrer un *service worker* à l'aide de la méthode `ServiceWorkerContainer.register()`. Si cela fonctionne, le *service worker* sera téléchargé sur le client et tentera les étapes d'installation et d'activation (voir ci-après) pour les URL auxquelles la personne accède pour toute l'origine concernée ou le sous-ensemble qui a été indiqué.

Téléchargement, installation et activation

À partir de ce point, le *service worker* suivra ce parcours :

1. Téléchargement
2. Installation
3. Activation

Le *service worker* est téléchargé immédiatement lorsque la personne accède pour la première fois à une page/un site contrôlé par un *service worker*.

Ensuite, le *service worker* est mis à jour :

- Lorsque la personne navigue sur une page concernée par la portée du *service worker*.
- Lorsqu'un évènement est déclenché sur le *service worker* et qu'il n'a pas été téléchargé lors des dernières 24 heures.

Une tentative d'installation a lieu lorsque le fichier téléchargé est nouveau, soit parce qu'il est différent (en termes d'octets) du *service worker* actuel, ou parce que c'est la première fois qu'un *service worker* est rencontré pour cette page/ce site.

Si c'est la première fois qu'un *service worker* est disponible, une tentative d'installation a lieu et si elle réussit, il est activé.

S'il y a déjà un *service worker* existant disponible, la nouvelle version est installée en arrière-plan mais n'est pas activée immédiatement. À cet instant, le *service worker* est considéré comme *worker en attente*. L'activation a lieu dès qu'il n'y a plus de pages chargées qui utilisent encore l'ancien *service worker*. Dès qu'il n'y a plus de pages à charger, le nouveau *service worker* est activé (devient donc le *worker actif*). L'activation peut être déclenchée plus tôt en utilisant `ServiceWorkerGlobalScope.skipWaiting()` (angl.) et les pages existantes peuvent alors être revendiquées par le *worker* actif avec `Clients.claim()`.

Il est possible d'écouter l'évènement `install` (angl.) ; cela permet généralement de préparer le *service worker* à être utilisé lorsque cet évènement se déclenche (par exemple en créant un cache avec l'API de stockage et en y plaçant des données qui permettront l'exécution de l'application hors-ligne).

Il existe également un évènement `activate` (angl.) qui est déclenché à l'activation. À ce moment, il est généralement utile de rafraîchir les vieux caches et autres éléments associés à l'ancienne version du *service worker*.

Un *service worker* peut répondre aux requêtes en utilisant l'évènement `fetch`. Les réponses à ces requêtes peuvent être modifiées en utilisant la méthode `FetchEvent.respondWith()` (angl.).

> **Note :** Comme les évènements `install` / `activate` peuvent prendre un certain temps à finir, la spécification fournit une méthode `waitUntil()` (angl.). Lorsque celle-ci est appelée sur les évènements `install` ou `activate` avec une promesse, les évènements fonctionnels comme `fetch` et `push` attendront la résolution de la promesse.

Pour un tutoriel complet illustrant comment construire un premier exemple simple fonctionnel, voir le guide Utiliser les *service workers*.

Utiliser le routage statique pour contrôler la façon dont les ressources sont récupérées

Les *service workers* peuvent entraîner un coût de performance inutile — lorsqu'une page est chargée pour la première fois depuis un certain temps, le navigateur doit attendre que le *service worker* démarre et s'exécute pour savoir quel contenu charger et s'il doit venir d'un cache ou du réseau.

Si vous savez déjà à l'avance d'où certains contenus doivent être récupérés, vous pouvez contourner le *service worker* et récupérer les ressources immédiatement. La méthode `InstallEvent.addRoutes()` (angl.) peut être utilisée pour ce cas d'usage et plus encore.

Autres idées de cas d'usage

Les *service workers* sont également conçus pour répondre à ces scénarios :

- Synchronisation de données en arrière-plan.
- Réponse à des requêtes de ressource depuis d'autres origines.
- Réception de mises à jour de données coûteuses en calcul (par exemple la géolocalisation ou le gyroscope) afin que plusieurs pages puissent utiliser un même ensemble de données.
- Compilation et gestion de dépendances côté client à des fins de développement (par exemple CoffeeScript, less, modules CJS/AMD).
- Déclencheurs (*hooks*) pour des services d'arrière-plan.
- Gestion de modèles personnalisés selon le motif des URL.
- Améliorations des performances, par exemple, le préchargement de ressources dont l'utilisateur·ice aura probablement bientôt besoin, comme les prochaines images d'un album photo.
- Simulation d'API.

À l'avenir, les *service workers* pourront réaliser d'autres tâches qui rapprocheront la plateforme web des applications natives. D'autres spécifications peuvent déjà exploiter les contextes des *service workers*, par exemple :

- Synchronisation en arrière-plan (angl.) : Démarrer un service worker même lorsqu'il n'y a personne sur le site, afin que les caches puissent être mis à jour, etc.
- Réaction aux messages push : Démarrer un service worker pour envoyer un message aux utilisateur·ice·s afin de leur indiquer qu'un nouveau contenu est disponible.
- Réaction à une date et une heure données.
- Entrée dans une zone géographique donnée.

Interfaces

`Cache`

Représente le stockage pour la paire d'objets `Request` / `Response` mis en cache pendant la vie du `ServiceWorker`.

`CacheStorage`

Représente le stockage pour les objets `Cache`. Fournit un répertoire racine pour tous les caches nommés auxquels un `ServiceWorker` peut accéder et maintient la liste des noms des objets `Cache` correspondants.

`Clients`

Représente un conteneur d'une liste d'objets `Client` ; il s'agit de la méthode principale pour accéder aux clients actifs du *service worker* à l'origine courante.

`ExtendableEvent`

Étend la durée de vie des évènements `install` et `activate` émis sur la portée `ServiceWorkerGlobalScope` (angl.) pendant le cycle de vie d'un *service worker*. Cela permet de s'assurer que les évènements fonctionnels (comme `FetchEvent`) ne sont pas diffusés au `ServiceWorker` tant qu'il n'a pas mis à jour ses schémas de base de données, supprimé ses éléments de cache obsolètes, etc.

`ExtendableMessageEvent`

Un objet représentant l'évènement `message` (angl.) déclenché sur un *service worker* (lorsqu'un message de canal est reçu sur la portée `ServiceWorkerGlobalScope` (angl.) depuis un autre contexte). Permet d'étendre la durée de vie de tels évènements.

`FetchEvent`

Le paramètre passé au gestionnaire d'évènement `onfetch` (angl.). Représente une action de récupération diffusée sur la portée `ServiceWorkerGlobalScope` (angl.) d'un `ServiceWorker`. Contient des informations à propos de la requête et de la réponse associé. Fournit une méthode `FetchEvent.respondWith()` (angl.) qui permet de fournir une réponse arbitraire à la page contrôlée.

`InstallEvent` (angl.)

Le paramètre passé au gestionnaire d'évènement `install` (angl.). Représente une action d'installation diffusée sur la portée `ServiceWorkerGlobalScope` (angl.) d'un `ServiceWorker`. Il s'agit d'une interface enfant de `ExtendableEvent` et permet donc de s'assurer que les évènements fonctionnels comme `FetchEvent` ne sont pas diffusés pendant l'installation.

`NavigationPreloadManager` (angl.)

Fournit des méthodes pour gérer le préchargement des ressources avec un *service worker*.

`ServiceWorker`

Représente un *service worker*. Plusieurs contextes de navigation (par exemple, des pages, des *workers*, etc.) peuvent être associés au même objet `ServiceWorker`.

`ServiceWorkerContainer`

Fournit un objet représentant le *service worker* comme une unité dans l'écosystème réseau, avec des utilitaires pour enregistrer, désenregistrer, mettre à jour des *service workers* et accéder à l'état des *service workers* et de leur enregistrement.

`ServiceWorkerGlobalScope` (angl.)

Représente le contexte global d'exécution d'un *service worker*.

`ServiceWorkerRegistration` (angl.)

Représente l'enregistrement d'un *service worker*.

`WindowClient`

Représente la portée d'un client de *service worker* qui est un document dans le contexte d'un navigateur, contrôlé par un *worker* actif. Il s'agit d'un type particulier d'objet `Client` avec certaines méthodes et propriétés complémentaires.

Extensions à d'autres interfaces

`Window.caches` et `WorkerGlobalScope.caches` (angl.)

Retourne l'objet `CacheStorage` associé au contexte courant.

`Navigator.serviceWorker` et `WorkerNavigator.serviceWorker` (angl.)

Retourne un objet `ServiceWorkerContainer`, qui donne accès à l'enregistrement, la suppression, la mise à jour et la communication avec les objets `ServiceWorker` pour le document associé (angl.).

Spécifications

Spécification
Service Workers Nightly

Voir aussi

- Utiliser les *Service Workers*
- Cycle de vie des *Service Workers* (angl.)
- Exemple de code de base pour les *Service Workers* (angl.)
- Les Web APIs qui sont liées à l'API *Service Worker* :
  - L'API de récupération en arrière-plan (angl.)
  - L'API de synchronisation en arrière-plan (angl.)
  - L'API d'indexation de contenu (angl.)

- L'API de stockage des cookies (angl.)
- L'API de notifications
- L'API de gestion des paiements basée sur le web (angl.)
- L'API de notifications push
- L'API de synchronisation périodique en arrière-plan (angl.)