Cache

Baseline Large disponibilité

Expérimental: Il s'agit d'une technologie expérimentale.
Vérifiez attentivement le tableau de compatibilité des navigateurs avant de l'utiliser en production.

L'interface `Cache` fournit un mécanisme de stockage pour les paires d'objets `Request`/`Response` qui sont mises en cache, par exemple dans le cadre du cycle de vie `ServiceWorker`. Il est à noter que l'interface `Cache` est exposée à des portées fenêtrées ainsi qu'à des service workers. Vous n'êtes pas obligé de l'utiliser avec des services workers, même si elle est définie dans la spécification relative aux services workers.

Une origine peut avoir plusieurs objets nommés `Cache`. Vous êtes responsable de l'implémentation de la manière dont votre script (par exemple dans un `ServiceWorker`) gère les mises à jour du cache. Les éléments d'un cache ne sont pas mis à jour, sauf demande explicite ; ils n'expirent pas, sauf s'ils sont supprimés. Utilisez `CacheStorage.open()` pour ouvrir un objet `Cache` spécifique et appelez ensuite l'une des méthodes `Cache` pour maintenir le `Cache`.

Vous êtes également responsable de la purge périodique des entrées du cache. Chaque navigateur a une limite stricte sur la quantité de mémoire cache qu'une origine donnée peut utiliser. Les estimations de l'utilisation du quota de cache sont disponibles via le lien `StorageEstimate` API. Le navigateur fait de son mieux pour gérer l'espace disque, mais il peut supprimer le stockage en cache d'une origine. Le navigateur supprime généralement toutes les données d'une origine ou aucune des données d'une origine. Veillez à nommer les caches et à n'utiliser les caches qu'à partir de la version du script sur laquelle ils peuvent fonctionner en toute sécurité. Pour plus d'informations, voir Suppression des anciens caches.

Note : Les implémentations initiales du cache (dans Blink et Gecko) résolvent les engagements `Cache.add()`, `Cache.addAll()`, et `Cache.put()` lorsque le corps de la réponse est entièrement écrit sur le stockage. Des versions plus récentes de la spécification précisent que le navigateur peut résoudre la promesse dès que l'entrée est enregistrée dans la base de données, même si le corps de réponse est encore en cours d'écriture.

Note : L'algorithme des correspondances de clés est dépendant de la valeur de l'en-tête VARY. Ainsi, pour faire correspondre une nouvelle clé, il faut examiner à la fois la clé et la valeur des entrées dans le Cache.

> **Note :** L'API de mise en cache n'honore pas les en-têtes de mise en cache HTTP.

Méthodes

`Cache.match(request, options)`

Retourne une `Promesse` qui se résout en une réponse associée à la première requête correspondante dans l'objet `Cache`.

`Cache.matchAll(request, options)`

Retourne une `Promesse` qui se résout en un tableau de toutes les requêtes correspondantes dans l'objet `Cache`.

`Cache.add(request)`

Prend une URL, la récupère et ajoute l'objet réponse associé au cache donné. C'est une fonctionnalité équivalente à l'appel de fetch(), puis à l'utilisation de Cache.put() pour ajouter les résultats au cache.

`Cache.addAll(requests)`

Prend un tableau d'URLs, les récupère, et ajoute les objets réponses associés au cache donné.

`Cache.put(request, response)`

Prend à la fois une requête et sa réponse et l'ajoute au cache donné.

`Cache.delete(request, options)`

Trouve l'entrée `Cache` dont la clé est la requête, et le cas échéant, supprime l'entrée `Cache` et retourne une `Promesse` qui se résout à `true`. Si aucune entrée `Cache` n'est trouvée, elle retourne `false`.

`Cache.keys(request, options)`

Retourne une `Promesse` qui se résout en un tableau clés `Cache`.

Exemples

Cet extrait de code provient de l'exemple de mise en cache sélective. (voir selective caching live) Le code utilise `CacheStorage.open(cacheName)` pour ouvrir tous les objets `Cache` avec un en-tête Content-Type qui débute par `font/`.

Le code utilise alors `Cache.match(request, options)` pour voir s'il y a déjà une fonte correspondante dans le cache, et le cas échéant, la retourne. S'il n'y a pas de correspondance, le code récupère la fonte à partir du réseau et utilise `Cache.put(request, response)` pour mettre en cache la ressource récupérée.

Le code gère les exceptions déclenchées par l'opération de `fetch()`. A noter qu'une réponse HTTP en erreur (e.g., 404) ne déclenchera pas une exception. Elle retournera un objet de réponse normal qui dispose du code d'erreur approprié.

Cet extrait de code illustre également une bonne pratique pour versionner les caches utilisés par le service worker. Bien qu'il y ait seulement un cache dans cet exemple, la même approche peut être utilisée pour des caches multiples. Il associe un identifiant court avec un nom de cache versionné et spécifique. Le code supprime aussi tous les caches qui ne sont pas nommés dans `CURRENT_CACHES`.

Note : In Chrome, visit chrome://inspect/#service-workers and click on the "inspect" link below the registered service worker to view logging statements for the various actions the service-worker.js script is performing.

JS

```
var CACHE_VERSION = 1;

// Shorthand identifier mapped to specific versioned cache.
var CURRENT_CACHES = {
  font: "font-cache-v" + CACHE_VERSION,
};

self.addEventListener("activate", function (event) {
  var expectedCacheNames = Object.keys(CURRENT_CACHES).map(function (key) {
    return CURRENT_CACHES[key];
  });

  // Active worker won't be treated as activated until promise resolves successfully.
  event.waitUntil(
    caches.keys().then(function (cacheNames) {
      return Promise.all(
        cacheNames.map(function (cacheName) {
          if (expectedCacheNames.indexOf(cacheName) == -1) {
            console.log("Deleting out of date cache:", cacheName);

            return caches.delete(cacheName);
          }
        }),
      );
    }),
  );
});

self.addEventListener("fetch", function (event) {
  console.log("Handling fetch event for", event.request.url);

  event.respondWith(
    // Opens Cache objects that start with 'font'.
    caches.open(CURRENT_CACHES["font"]).then(function (cache) {
      return cache
        .match(event.request)
        .then(function (response) {
          if (response) {
            console.log(" Found response in cache:", response);
```

```
            return response;
          }
        })
        .catch(function (error) {
          // Handles exceptions that arise from match() or fetch().
          console.error("  Error in fetch handler:", error);

          throw error;
        });
    }),
  );
});
```

Storing cookies in Caches

L'API Fetch exige que les en-têtes `Set-Cookie` soient supprimés avant de renvoyer un objet `Response` à partir de `fetch()`. Ainsi, une réponse stockée dans un cache ne contiendra pas d'en-têtes.

Spécifications

Spécification
Service Workers Nightly # cache-interface

Compatibilité des navigateurs

Signaler des problèmes avec ces données de compatibilité • Voir les données sur GitHub

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Browser	WebView Android	WebView on iOS	Deno
`Cache`	✓ 40 *	✓ 16	✓ 41	✓ 27 *	✓ 11.1	✓ 40 *	✓ 41	✓ 27 *	✓ 11.3	✓ 4	✓ 40 *	✓ 11.3	✓ 1.26
add	✓ 44 *	✓ 16	✓ 41	✓ 31 *	✓ 11.1	✓ 44 *	✓ 41	✓ 32 *	✓ 11.3	✓ 4 *	✓ 44 *	✓ 11.3	⊗ Non

`addAll`	✓ 46 *	✓ 16	✓ 41	✓ 33 *	✓ 11.1	✓ 46 *	✓ 41	✓ 33 *	✓ 11.3	✓ 5 *	✓ 46 *	✓ 11.3	⊗ Non
`delete`	✓ 43	✓ 16	✓ 41	✓ 30	✓ 11.1	✓ 43	✓ 41	✓ 30	✓ 11.3	✓ 4	✓ 43	✓ 11.3	✓ 1.26 *
`keys`	✓ 43	✓ 16	✓ 41	✓ 30	✓ 11.1	✓ 43	✓ 41	✓ 30	✓ 11.3	✓ 4	✓ 43	✓ 11.3	✓ 2.7.14
`match`	✓ 43	✓ 16	✓ 41	✓ 30	✓ 11.1	✓ 43	✓ 41	✓ 30	✓ 11.3	✓ 4	✓ 43	✓ 11.3	✓ 1.26 *
`matchAll`	✓ 47	✓ 16	✓ 41	✓ 34 *	✓ 11.1	✓ 47	✓ 41	✓ 34	✓ 11.3	✓ 5	✓ 47	✓ 11.3	⊗ Non
`put`	✓ 43 *	✓ 16	✓ 41	✓ 30 *	✓ 11.1	✓ 43 *	✓ 41	✓ 30 *	✓ 11.3	✓ 4 *	✓ 43 *	✓ 11.3	✓ 1.26
Available in workers	✓ 40 *	✓ 16	✓ 44	✓ 27 *	✓ 11.1	✓ 40 *	✓ 44	✓ 27 *	✓ 11.3	✓ 4	✓ 40 *	✓ 11.3	✓ 1.26

✓ Prise en charge complète ⊗ Pas de prise en charge * Voir les notes de mise en application.

Voir aussi

- Utiliser les Service Workers
- Code d'exemple basique de Service workers
- Le ServiceWorker est prêt ?
- `Promesse`
- Utilisation des Web Workers