19

Respuesta a: **Service Workers**

AugdiAugus 8186

Service Worker API

Los Service workers actúan esencialmente como proxy servers asentados entre las aplicaciones web, el navegador y la red (cuando está accesible). Están destinados, entre otras cosas, a permitir la creación de experiencias offline efectivas, interceptando peticiones de red y realizando la acción apropiada si la conexión de red está disponible y hay disponibles contenidos actualizados en el servidor. También permitirán el acceso a notificaciones tipo push y APIs de sincronización en segundo plano.

Descarga, instalación y activación

En este punto, su service worker observará el siguiente ciclo de vida:

1. Descarga
2. Instalación
3. Activación

El service worker se descaga inmediatamente cuando un usuario accede por primera vez a un sitio controlado por el mismo.
Después de esto se descarga cada 24 horas aproximadamente. Se puede descargar con más frecuencia, pero **debe** ser descargado cada 24 horas para prevenir que una mala programación sea molesta durante mucho tiempo.

ServiceWorkerContainer.register()

Parámetros

```
ServiceWorkerContainer.register(scriptURL, options)
 .then(function(ServiceWorkerRegistration) { ... });
```

`scriptURL`
La URL del script de trabajador de servicio.
`optionsOptional`
Un objeto que contiene opciones de registro. Las opciones disponibles actualmente son:
`alcance`: `USVString` representa una URL que define el alcance de registro de un “service worker”; es decir, qué rango de URL puede controlar un “service worker”. Esto es generalmente una URL relativa. El valor predeterminado es la URL que obtendría si resolviera ‘./’ utilizando la ubicación de la página web como base. No es, como se cree comúnmente, relativo a la ubicación del “service worker”. Vea la sección de Ejemplos para más información sobre cómo funciona.

Ejemplo

Ejemplo

```
if ('serviceWorker'in navigator) {
  // Register a service worker hosted at the root of the
  // site using the default scope.
  navigator.serviceWorker.register('/sw.js').then(function(registration) {
    console.log('Service worker registration succeeded:', registration);
  }).catch(function(error) {
    console.log('Service worker registration failed:', error);
  });
} else {
  console.log('Service workers are not supported.');
}
```

CacheStorage.open()

The **`open()`** method of the `CacheStorage` interface returns a `Promise` that resolves to the `Cache` object matching the `cacheName`.

Ejemplo

```
var cachedResponse = caches.match(event.request).catch(function() {
  return fetch(event.request);
}).then(function(response) {
  caches.open('v1').then(function(cache) {
    cache.put(event.request, response);
  });
  return response.clone();
}).catch(function() {
  return caches.match('/sw-test/gallery/myLittleVader.jpg');
});
```

Cache

La **`Cache`**interfaz proporciona un mecanismo de almacenamiento para `[Request](http://fetch.spec.whatwg.org/#request)`/`[Response](http://fetch.spec.whatwg.org/#response)` pares de objetos que se almacenan en caché, por ejemplo, como parte del `ServiceWorker`ciclo de vida. Tenga en cuenta que la `Cache`interfaz está expuesta a ámbitos con ventanas, así como a los trabajadores. No tiene que usarlo junto con los trabajadores del servicio, aunque esté definido en la especificación del trabajador del servicio.

MétodosSección

`Cache.match(request, options)`

Devuelve un `Promise`que resuelve la respuesta asociada con la primera solicitud coincidente en el `Cache`objeto.

`Cache.matchAll(request, options)`

Devuelve un `Promise`que resuelve una matriz de todas las solicitudes coincidentes en el `Cache`objeto.

`Cache.add(request)`
Toma una URL, la recupera y agrega el objeto de respuesta resultante a la caché dada. Esto es funcionalmente equivalente a llamar `fetch()`, luego usar `put()` para agregar los resultados al caché.

`Cache.addAll(requests)`
Toma una matriz de URL, las recupera y agrega los objetos de respuesta resultantes a la caché dada.
`Cache.put(request, response)`
Toma tanto una solicitud como su respuesta y la agrega al caché dado.
`Cache.delete(request, options)`
Encuentra la `Cache` entrada cuya clave es la solicitud, devolviendo una `Promise`que se resuelve `true`si `Cache`se encuentra y elimina una entrada coincidente . Si no `Cache`se encuentra ninguna entrada, la promesa se resuelve `false`.
`Cache.keys(request, options)`
Devuelve un `Promise`que se resuelve en una matriz de `Cache`claves.

0 hace 9 meses

Frontend con React.JS

Curso Profesional de JavaScript • Service Workers

Escribe tu comentario

+ 2