

Imagina que estás en un tren usando tu teléfono móvil para navegar por tu sitio web favorito.

Cada vez que el tren ingresa a un área con una red poco confiable, el sitio web tarda mucho en cargarse, una escena muy familiar. Aquí es donde el almacenamiento en caché de Service Worker viene al rescate. El almacenamiento en caché garantiza que su sitio web se cargue de la manera más eficiente posible para los visitantes habituales.

Este capítulo comienza analizando los conceptos básicos del almacenamiento en caché HTTP y lo que sucede internamente cuando su navegador navega a una URL. También veremos detenidamente cómo puede utilizar el almacenamiento en caché de Service Worker para proporcionar a sus usuarios un sitio web más rápido y confiable y cómo funciona de la mano con el almacenamiento en caché HTTP tradicional.

Aprenderá cómo puede utilizar el almacenamiento en caché de Service Worker en una aplicación del mundo real, incluidos los recursos de control de versiones y almacenamiento en caché previo. Finalmente, descubrirá una de mis bibliotecas favoritas de Service Worker: Workbox.

3.1 Los conceptos básicos del almacenamiento en caché HTTP

Los navegadores modernos son inteligentes. Pueden interpretar y comprender una variedad de solicitudes y respuestas HTTP y son capaces de almacenar y almacenar en caché datos hasta que sean necesarios. Me gusta pensar en la capacidad del navegador para almacenar información en caché como la fecha de caducidad de la leche. De la misma manera que puede guardar leche en su refrigerador hasta que llegue a la fecha de caducidad, los navegadores pueden almacenar en caché información sobre un sitio web durante un período de tiempo determinado. Una vez que los datos hayan caducado, buscará la versión actualizada. Esto garantiza que las páginas web se carguen más rápido y utilicen menos ancho de banda.

Antes de sumergirnos en el almacenamiento en caché de Service Worker, retrocedamos un paso y veamos cómo funciona el almacenamiento en caché HTTP tradicional. Los desarrolladores web han podido utilizar HTTP

almacenamiento en caché desde la introducción de HTTP/1.0 a principios de la década de 1990.¹ Almacenamiento en caché de HTTP permite al servidor enviar los encabezados HTTP correctos que le indicarán al navegador que almacenar en caché la respuesta durante un cierto período de tiempo.

Un servidor web puede aprovechar la capacidad del navegador para almacenar datos en caché y utilizarlos para mejorar el tiempo de carga de solicitudes repetidas. Si el usuario visita la misma página dos veces dentro de un sesión, a menudo no hay necesidad de ofrecerles una versión nueva de los recursos si los datos no ha cambiado. De esta manera, un servidor web puede usar el encabezado Expires para notificar a la web cliente que puede usar la copia actual de un recurso hasta la "fecha de vencimiento" especificada. A su vez, el navegador puede almacenar en caché este recurso y solo volver a buscar una nueva versión. cuando llegue a la fecha de vencimiento. La Figura 3.1 ilustra el almacenamiento en caché HTTP.

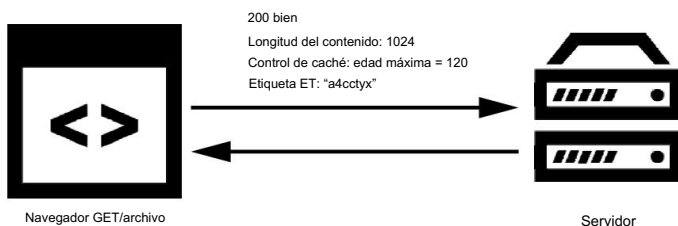


Figura 3.1 Cuando un navegador realiza una solicitud HTTP para un recurso, el servidor envía una respuesta HTTP que contiene información útil sobre el recurso.

En la figura 3.1, puede ver que cuando un navegador realiza una solicitud de un recurso, el servidor devuelve el recurso con una colección de encabezados HTTP. Estos encabezados contienen información útil que el navegador puede utilizar para comprender más sobre el recurso. La respuesta HTTP le dice al navegador qué tipo de recurso es, cuánto tiempo para almacenarlo en caché, si está comprimido y mucho más.

El almacenamiento en caché HTTP es una manera fantástica de mejorar el rendimiento de su sitio web, pero no está exento de defectos. Usar el almacenamiento en caché HTTP significa que confías en el servidor para saber usted sabe cuándo almacenar en caché un recurso y cuándo caduca. Si tiene contenido que tiene dependencias, cualquier actualización puede hacer que las fechas de vencimiento enviadas por el servidor se eliminen fácilmente. de sincronización y afectar su sitio.

Un gran poder conlleva una gran responsabilidad, y esto es muy cierto en el caso del almacenamiento en caché HTTP. Cuando realiza cambios significativos en HTML, es probable que también cambie el CSS para reflejar la nueva estructura y actualizar cualquier JavaScript para adaptarse a los cambios. al estilo y al contenido. Si alguna vez ha publicado cambios en un sitio web pero aún no lo ha hecho Si tienes el caché HTTP correcto, estoy seguro de que has visto que el sitio web se rompió debido a recursos almacenados en caché incorrectamente.

¹ <https://hpbnc.co/breve-historia-de-http/>

La Figura 3.2 muestra cómo se ve mi blog personal cuando tengo archivos almacenados en caché incorrectamente.

Como puede imaginar, esto puede resultar bastante frustrante tanto para el desarrollador como para el usuario. En la figura 3.2, puede ver que los estilos CSS de la página no se cargan. Esto se debe a que el almacenamiento en caché incorrecto provocó una discrepancia.

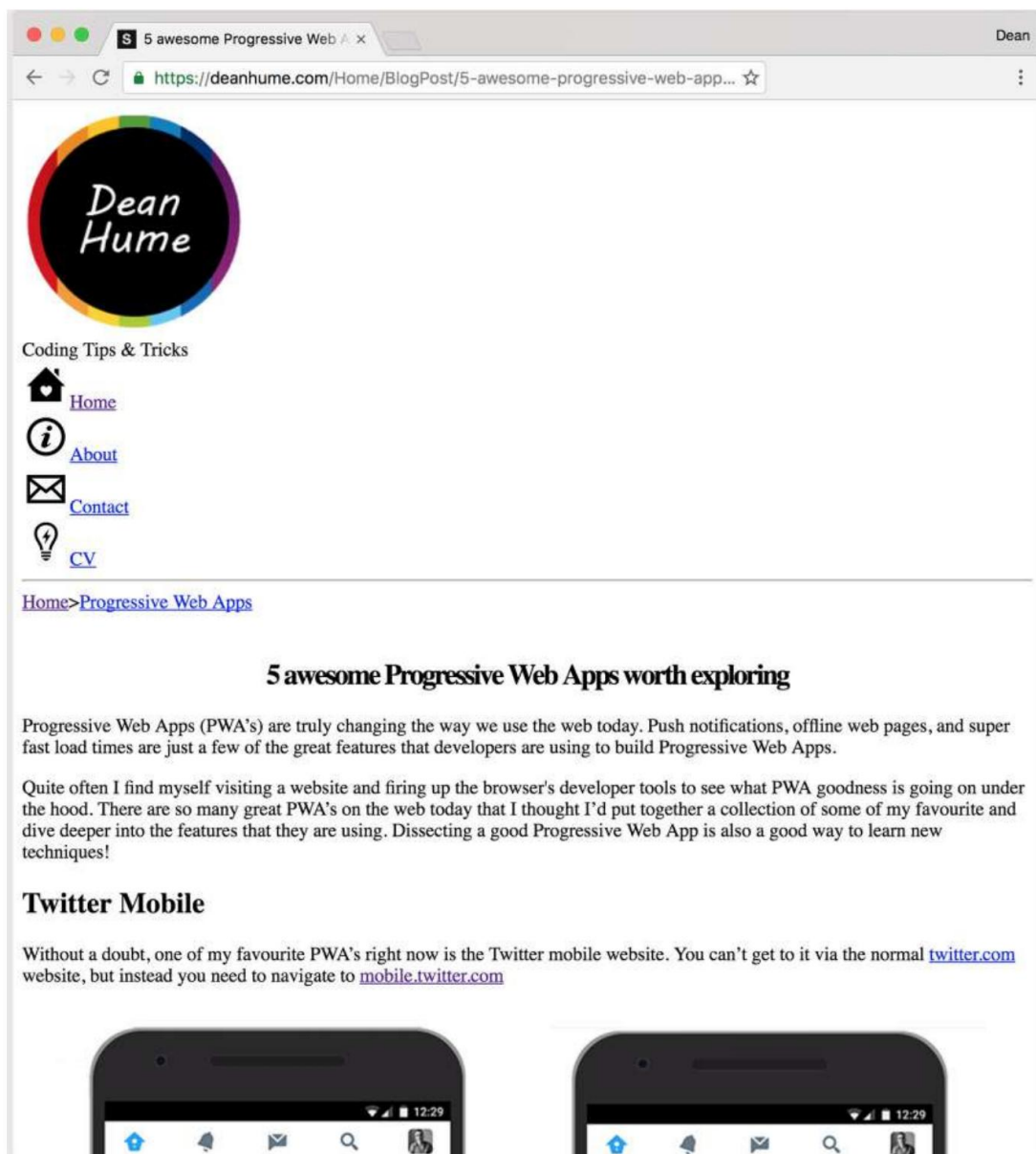


Figura 3.2 Cuando los archivos almacenados en caché no están sincronizados, la apariencia de su sitio web puede verse afectada.

3.2 Los conceptos básicos del almacenamiento en caché Almacenamiento en caché del trabajador del servicio

Quizás se pregunte por qué necesita el almacenamiento en caché de Service Worker si tiene almacenamiento en caché HTTP. ¿En qué se diferencia el almacenamiento en caché de Service Worker? Bueno, en lugar de que el servidor le diga al navegador cuánto tiempo debe almacenar en caché un recurso, usted tiene el control total. El almacenamiento en caché de Service Worker es extremadamente poderoso porque le brinda control programático sobre exactamente cómo almacena en caché sus recursos. Al igual que con todas las funciones de la aplicación web progresiva (PWA), el almacenamiento en caché de Service Worker es una mejora del almacenamiento en caché HTTP y funciona de la mano con él.

El poder de los Service Workers radica en su capacidad para interceptar solicitudes HTTP. En este capítulo, utilizará esta capacidad para interceptar solicitudes y respuestas HTTP para proporcionar a los usuarios una respuesta ultrarrápida directamente desde la memoria caché.

3.2.1 Almacenamiento en caché previo durante la instalación del Service Worker

Con Service Workers, puede acceder a cualquier solicitud HTTP entrante y decidir exactamente cómo desea responder. En su Service Worker, puede escribir lógica para decidir qué recursos desea almacenar en caché, qué condiciones deben cumplirse y durante cuánto tiempo almacenar en caché un recurso. Estas en total control.

Quizás esté familiarizado con la figura 3.3; la analizamos brevemente en capítulos anteriores de este libro. Cuando el usuario visita el sitio web por primera vez, el Service Worker comienza a descargarse e instalarse. Durante la etapa de instalación, puede aprovechar este evento y preparar el caché con todos los activos críticos para la aplicación web.

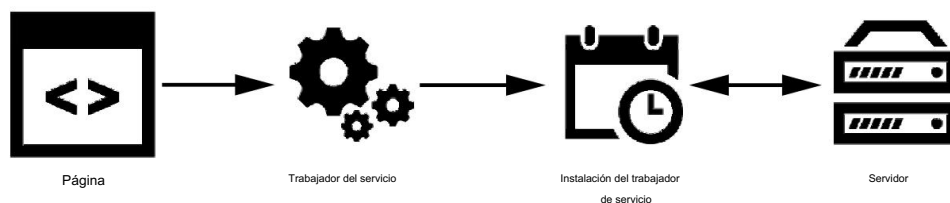


Figura 3.3 Durante el paso de instalación de Service Worker, puede recuperar recursos y preparar el caché para la próxima visita.

Usando esta figura como ejemplo, veamos un ejemplo básico de almacenamiento en caché para comprender mejor cómo funciona esto en la realidad. La siguiente lista muestra una página HTML simple que registra un archivo de Service Worker.

Listado 3.1 Página HTML simple que registra un archivo Service Worker

```

<!DOCTYPE html>
<html>
  <cabeza>
    <meta charset="UTF-8"> <title>¡Hola
    mundo del almacenamiento en caché!</title>
  </cabeza>
  < cuerpo>
    <script src="sw.js"></script>
  </cuerpo>
</html>
  
```

```
</cabeza>
< cuerpo>
  <!-- Imagen -->
   <!-- JavaScript -->

  <script async src="/js/script.js"></script> <script>

// Registre el trabajador del servicio if ('serviceWorker' en el navegador) { navigator.serviceWorker.register('/
  service-worker.js').then(function(registration) {

    // El registro fue exitoso
    console.log('Registro de ServiceWorker exitoso con alcance: ', registro.scope);

  }).catch(function(err) { // error de registro :
    (
      console.log('Error en el registro de ServiceWorker: ', err);
    );
  }
  </script>
</ cuerpo>
</html>
```

Referencia a una imagen de "hola"

Referencia a un archivo JavaScript básico

Verifique si el navegador actual admite Service Workers.

Si se produce un error durante el servicio de trabajador registro, puedes atraparlo y responder apropiadamente

En el listado 3.1, ve una página web simple que hace referencia a una imagen y un archivo JavaScript. La página web no es nada sofisticada, pero la usarás para aprender a almacenar recursos en caché. utilizando el almacenamiento en caché de Service Worker. El código comprueba si su navegador admite Service Workers; Si es así, intentará registrar un archivo llamado service-worker.js, suponiendo que esté siguiendo el juego en casa.

Ya tenemos lista nuestra página básica. A continuación necesitas crear el código que almacenará en caché tus recursos. El código del siguiente listado va dentro del archivo del trabajador del servicio. trabajador-de-servicio.js.

Listado 3.2 Código en service-worker.js

```
var cacheName = 'holaMundo';

self.addEventListener('instalar', evento => {
  evento.esperar hasta (
    caches.open(cacheName) .luego(cache
      => cache.addAll([
        '/js/script.js',
        '/images/hola.png'
      ])
    );
  });
});
```

Nombre del caché

Acceda al servicio
Evento de instalación de trabajadores

Abra un caché usando el nombre del caché que especificamos

Agregue JavaScript y la imagen al caché

En el capítulo 1, analizamos el ciclo de vida del Service Worker y las diferentes etapas que pasa antes de que se vuelva activo. Una de estas etapas es el evento de instalación, que Ocurre cuando el navegador instala y registra el Service Worker. Este es el momento perfecto para agregar al caché cualquier cosa que crea que podría usarse en una etapa posterior.

Por ejemplo, si sabe que un archivo JavaScript específico podría usarse en todo el sitio, puede decidir almacenarlo en caché durante la instalación. Eso significaría que cualquier otras páginas que hacen referencia a este archivo JavaScript podrán recuperarlo fácilmente desde caché en una etapa posterior.

El código del listado 3.2 aprovecha el evento de instalación y agrega el archivo JavaScript y la imagen de saludo durante esta etapa. También hace referencia a una variable llamada `cacheName`. Este es un valor de cadena que configuré para nombrar el caché. Puedes nombrar cada caché de manera diferente e incluso puedes tener múltiples copias diferentes del caché porque cada nueva cadena lo hace único. Esto será útil más adelante en el capítulo, cuando analicemos el control de versiones y la eliminación de caché.

En el listado 3.2, puede ver que una vez que se ha abierto el caché, puede comenzar a agregarle recursos. Luego llamas a `cache.addAll()` y pasas tu matriz de archivos. El método `event.waitUntil()` utiliza una promesa de JavaScript para saber cuánto tiempo tarda la instalación y si se realizó correctamente.

Si todos los archivos se almacenan en caché correctamente, se instalará Service Worker. Si alguno de los archivos no se descargan, el paso de instalación fallará. Esto es importante porque significa que debe confiar en que todos los activos estén presentes en el servidor y debe tener cuidado con la lista de archivos que decide almacenar en caché en el paso de instalación. Definiendo una lista larga de archivos aumentará las posibilidades de que un archivo no se almacene en caché, lo que provocará su Service Worker no está instalado.

Ahora que su caché está preparado y listo para funcionar, puede comenzar a leer recursos. De eso. Debe agregar el código en el siguiente listado a su trabajador de servicio para poder comenzar a escuchar el evento de recuperación.

Listado 3.3 Código para agregar al Service Worker para comenzar a escuchar el evento de recuperación

```
self.addEventListener('buscar', función(evento) {
  evento.responderCon(
    cachés.match(evento.solicitud)
      .entonces(función(respuesta) {
        si (respuesta) {
          respuesta de devolución;
        }
        retorno de recuperación (evento.solicitud); #MI
      })
  );
});
```

Agregue un detector de eventos al evento de recuperación

Compruebe si la URL de solicitud entrante coincide con algo que exista en el caché actual

Si hay una respuesta y no es indefinida/nula, devuélvela

De lo contrario continúa con normalidad. y buscar el recurso

Como era la intención

El código del listado 3.3 es la pieza final de nuestra obra maestra de Service Worker. tu empiezas agregando un detector de eventos para el evento de recuperación. A continuación, comprueba si la URL entrante coincide con cualquier cosa que pueda existir en su caché actual usando `caches.match()` función. Si es así, devuelva ese recurso almacenado en caché, pero si el recurso no existe en caché, continúe normalmente y obtenga el recurso solicitado.

Si abre un navegador que admita Service Workers y navega a esta página recién creada, debería notar algo similar a la figura 3.4.

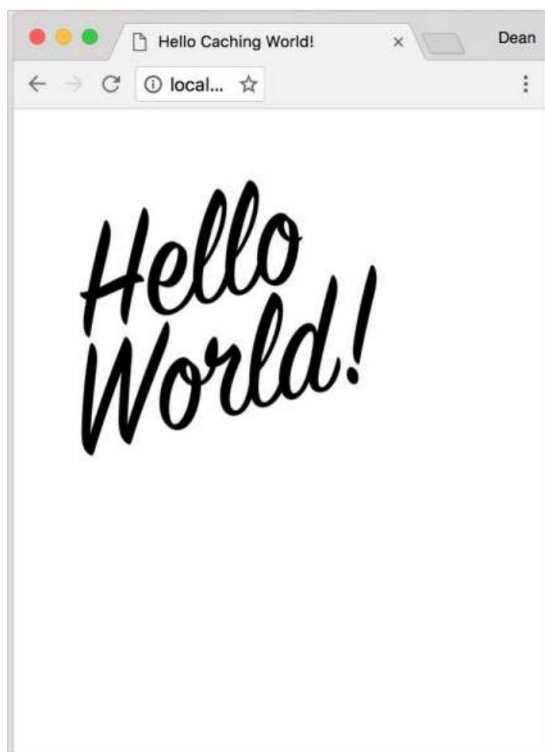


Figura 3.4 El código de muestra produce una página web básica con una imagen y un archivo JavaScript.

Los recursos solicitados ahora deberían estar disponibles en la caché del Service Worker. Cuando actualizo la página, el trabajador del servicio interceptará la solicitud HTTP y cargará los recursos apropiados instantáneamente desde el caché en lugar de realizar una solicitud de red al servidor. En unas pocas líneas de código dentro de un Service Worker, ha creado un sitio que se carga directamente desde la memoria caché y responde instantáneamente ante visitas repetidas.

NOTA Los trabajadores del servicio solo funcionan en orígenes seguros como HTTPS. Pero cuando desarrolla Service Workers en su máquina local, puede usar <http://localhost>. Los Service Workers se han creado de esta manera para garantizar la seguridad cuando se implementan en vivo, y también para brindar flexibilidad, para facilitar a los desarrolladores trabajar en su máquina local.

Algunos navegadores modernos pueden ver lo que hay dentro de la caché del Service Worker utilizando las herramientas de desarrollo integradas en el navegador. Por ejemplo, si abre las Herramientas para desarrolladores de Google Chrome y navega a la pestaña Aplicación, verá algo similar a la figura 3.5.

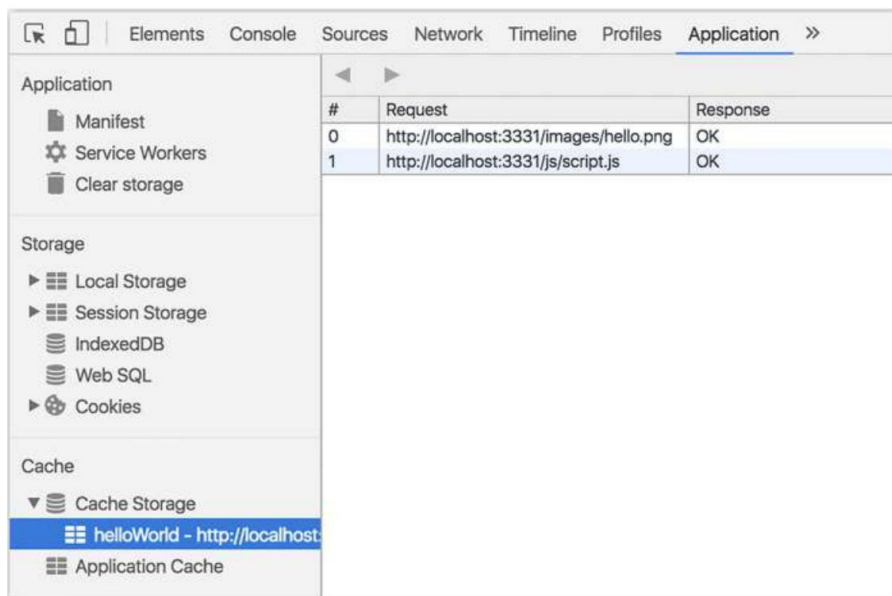


Figura 3.5 Las herramientas para desarrolladores de Google Chrome son útiles cuando desea ver lo que está almacenado en la memoria caché.

La Figura 3.5 muestra las entradas de la caché para los archivos `scripts.js` y `hello.png` almacenados en la caché denominada `helloWorld`. Ahora que los recursos se han almacenado en la memoria caché, cualquier solicitud futura de esos recursos se recuperará instantáneamente de la memoria caché.

3.2.2 Intercepción y caché

El Listado 3.2 mostró cómo se pueden almacenar en caché recursos importantes durante la instalación de un Service Worker, lo que se conoce como almacenamiento en caché previo. Este ejemplo funciona bien cuando sabes exactamente los recursos que deseas almacenar en caché, pero ¿qué pasa con los recursos que pueden ser dinámicos o que quizás no conoces? Por ejemplo, su sitio web podría ser un sitio web de noticias deportivas que necesite una actualización constante durante un partido; no sabrá acerca de esos archivos durante la instalación de Service Worker.

Debido a que los Service Workers pueden interceptar solicitudes HTTP, esta es la oportunidad perfecta para realizar la solicitud HTTP y luego almacenar la respuesta en caché. Esto significa que, en lugar de ello, solicita el recurso y luego lo almacena en caché inmediatamente. De esa manera, cuando se realice la siguiente solicitud HTTP para el mismo recurso, podrá recuperarlo instantáneamente de la caché de Service Worker, como se muestra en la figura 3.6.

La siguiente lista actualiza el código que utilizó anteriormente para incluir un nuevo recurso.

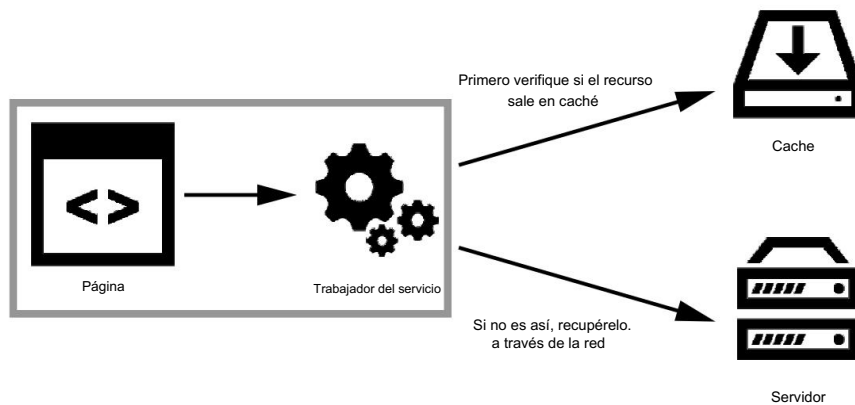


Figura 3.6 Para cualquier solicitud HTTP realizada, puede verificar si el recurso ya existe en la memoria caché y, en caso contrario, lo recuperamos a través de la red.

Listado 3.4 Una página web básica para mostrar fuentes de Google

```

<!DOCTYPE html>
<html>
  <cabeza>
    <meta juego de caracteres="UTF-8">
    <title>¡Hola, mundo del almacenamiento en caché!</title>
    <enlace href="https://fonts.googleapis.com/css?family=Lato"
      rel="hoja de estilo">
  </cabeza>
  <estilo>
    #body{ font-family: 'Lato', sans-serif; }
  </estilo>
  <cuerpo>
    <h1>¡Hola, caché de trabajador de servicio!</h1>
    <!-- JavaScript -->
    <script async src="/js/script.js"></script> <script>

if ('serviceWorker' en el navegador)
  { navigator.serviceWorker.register('/service-
    worker.js').entonces(función(registro) {
      console.log('Registro de ServiceWorker exitoso con alcance: ', registro.scope);

    }).catch(function(err) { console.log('Error
      en el registro de ServiceWorker: ', err);
    });
  }
</script>
</cuerpo>
</html>

```

← Agregue una referencia a fuentes web.

← Archivo JavaScript que proporciona funcionalidad para la página actual.

← Primero verifique si el navegador admite trabajadores de servicios.

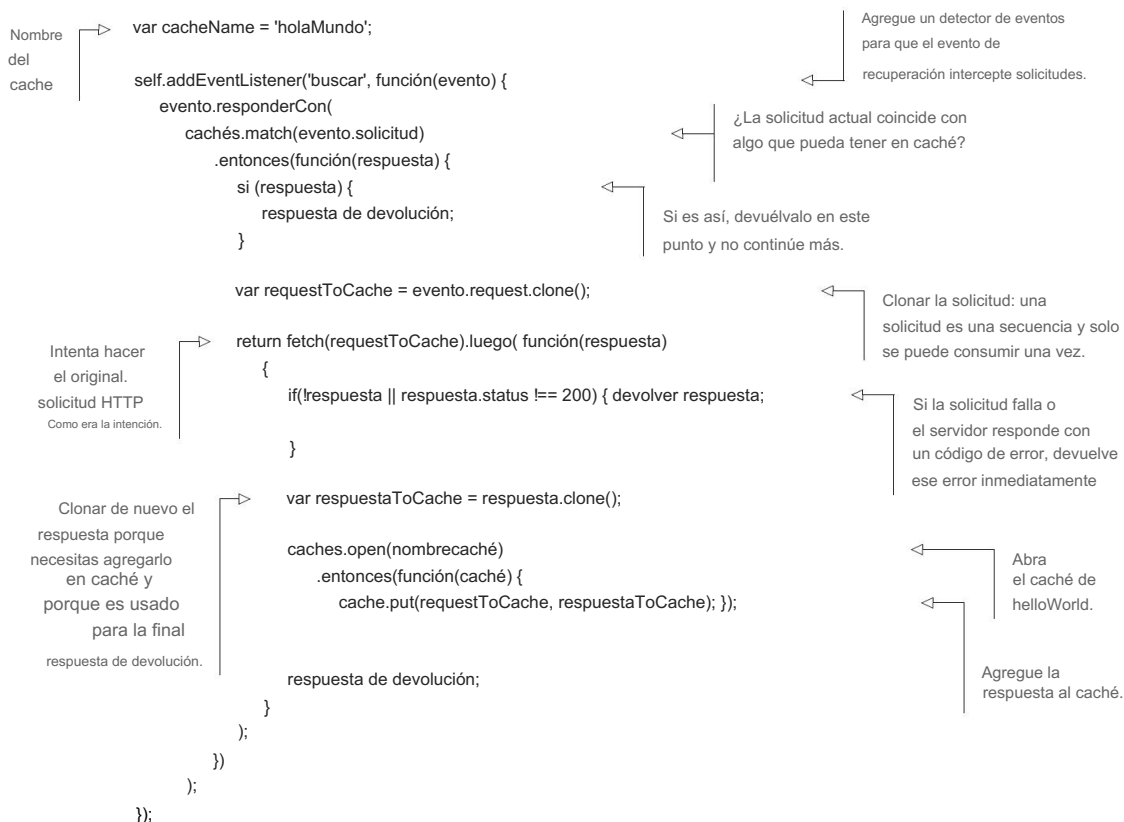
← Si hay un error durante el registro de trabajador de servicio, puede atraparlo y responder apropiadamente.

En el listado 3.4, el código no ha cambiado mucho en comparación con el listado 3.1, excepto que ha agregado una referencia a fuentes web en la etiqueta HEAD. porque esto es un extra recurso que probablemente cambie, puede almacenar en caché el recurso una vez que HTTP

se ha realizado la solicitud. También notarás que el código JavaScript utilizado para registrar el El trabajador de servicio no ha cambiado. De hecho, con algunas excepciones, este código es bastante forma estándar de registrar a su trabajador de servicio. Utilizarás este código repetitivo. para registrar un trabajador de servicio repetidamente a lo largo del libro.

Ahora que la página está completa, está listo para comenzar a agregar código al archivo de Service Worker. La siguiente lista muestra el código que utilizará.

Listado 3.5 Agregar código al archivo Service Worker



El Listado 3.5 parece mucho código. Desglosémoslo y expliquemos cada sección. El El código comienza accediendo al evento de recuperación agregando un detector de eventos. La primera Lo que desea hacer es verificar si el recurso solicitado ya existe en la memoria caché. Si es así, puedes devolverlo en este punto y no continuar.

Pero si el recurso solicitado aún no existe en el caché, realiza la solicitud como se pretendía originalmente. Antes de que el código avance, debemos clonar. la solicitud porque una solicitud es una secuencia que solo se puede consumir una vez. Porque estás consumiendo esto una vez por caché y luego otra vez cuando haces el HTTP Si lo solicita, debe clonar la respuesta en este punto. Entonces necesitas comprobar

la respuesta HTTP y asegúrese de que el servidor haya devuelto una respuesta exitosa y que nada haya salido mal. No desea almacenar en caché un resultado erróneo.

Si la respuesta fue exitosa, volverás a clonar la respuesta. Probablemente se esté preguntando por qué necesita clonar la respuesta nuevamente, pero recuerde que una respuesta es una secuencia que solo se puede consumir una vez. Debido a que desea que el navegador consuma la respuesta y que el caché consuma la respuesta, debe clonarlo para tener dos corrientes.

Finalmente, el código usa esta respuesta y la agrega al caché para que pueda usarla nuevamente la próxima vez. Si el usuario luego actualiza la página o visita otra página en el sitio que requiere estos recursos, se recuperará del caché instantáneamente en lugar de a través de la red.

En la figura 3.7, observe que hay nuevas entradas en el caché para los tres recursos de la página. En el ejemplo de codificación visto anteriormente, pudo agregar dinámicamente un recurso al caché a medida que se devolvía cada respuesta HTTP exitosa. Esta técnica es perfecta para cuando desee almacenar en caché recursos pero no esté seguro de con qué frecuencia pueden cambiar o de dónde pueden provenir exactamente.

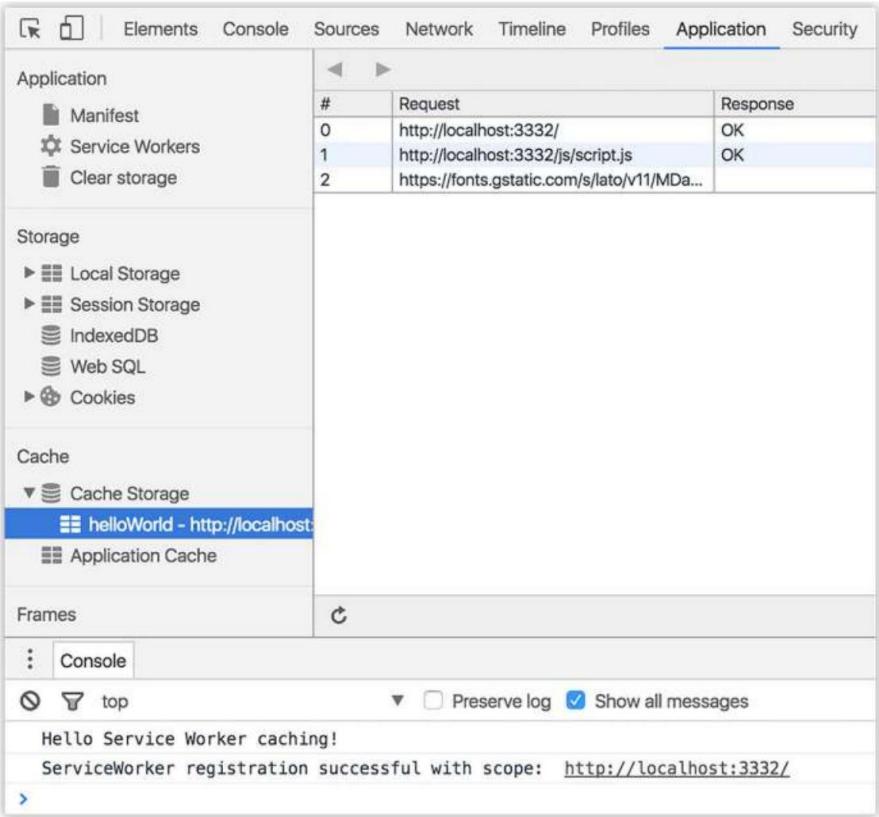


Figura 3.7 Al utilizar las herramientas para desarrolladores de Google Chrome, verá que las fuentes web se recuperaron de la red y luego se agregaron al caché para garantizar solicitudes repetidas más rápidas.

Los Service Workers brindan al desarrollador control total sobre el código y le permiten crear fácilmente soluciones de almacenamiento en caché personalizadas que se ajusten a sus necesidades. De hecho, las dos técnicas de almacenamiento en caché analizadas anteriormente se pueden combinar para producir tiempos de carga aún más rápidos. El control está en tus manos.

Por ejemplo, supongamos que está creando una nueva aplicación web que utiliza la arquitectura App Shell. Es posible que desee almacenar en caché el shell utilizando el código del listado 3.2.

Luego, cualquier solicitud HTTP adicional que se realice se puede almacenar en caché utilizando la técnica de intercepción y caché. O tal vez desee almacenar en caché partes de un sitio existente que sabe que no cambian con frecuencia. Al interceptar y almacenar en caché estos recursos, brindará a sus usuarios un rendimiento mejorado en unas pocas líneas de código. Dependiendo de su situación, el almacenamiento en caché de Service Worker se puede adaptar para satisfacer sus necesidades y marcar una diferencia instantánea en la experiencia que reciben sus usuarios.

3.2.3 Poniéndolo todo junto

Los ejemplos de código que hemos visto hasta ahora han sido útiles, pero no es fácil imaginarlos por sí solos. En el capítulo 1, hablamos sobre las diferentes formas en que se pueden utilizar Service Workers para crear aplicaciones web increíbles. Uno de esos conceptos fue una aplicación web de periódico, que podemos usar para jugar con todo lo que ha aprendido sobre el almacenamiento en caché de Service Worker en un escenario del mundo real. Voy a llamar a nuestra aplicación de muestra Tiempos Progresivos. La aplicación web es un sitio de noticias donde las personas visitan y leen varias páginas con regularidad, por lo que tiene sentido almacenar en caché las páginas futuras con anticipación para que se carguen instantáneamente. Incluso podría guardar el contenido para que un usuario pueda navegar sin conexión.

La aplicación web de muestra contiene una colección de noticias divertidas de todo el mundo (figura 3.8). Lo creas o no, todas las historias contenidas en este sitio de noticias son ciertas y provienen de fuentes de noticias creíbles. La aplicación web contiene la mayoría de los elementos básicos de un sitio web que puedas imaginar, como CSS, JavaScript e imágenes. Para mantener el código de muestra básico, también he usado un archivo JSON plano para cada artículo; en la vida real, esto apuntaría a un punto final de back-end para recuperar los datos en un formato similar. Por sí sola, esta aplicación web no es tan impresionante, pero cuando comienzas a utilizar el poder de los Service Workers, puedes llevarla al siguiente nivel.

La aplicación web utiliza la arquitectura App Shell para recuperar dinámicamente el contenido. carpas de cada artículo e inyectar los datos en la página, como se muestra en la figura 3.8.

El uso de la arquitectura App Shell también significa que puede utilizar el almacenamiento en caché previo para garantizar que la aplicación web se cargue instantáneamente para visitas repetidas. También puede suponer que un visitante tocará un enlace y seguirá el contenido completo de un artículo de noticias. Si almacenó esto en caché cuando se instaló Service Worker, significaría que la siguiente página se cargaría significativamente más rápido para ellos.

Juntemos todo lo que aprendió hasta ahora en el capítulo y veamos cómo agregar un trabajador de servicio a la aplicación Progressive Times que almacenará en caché los recursos importantes y almacenará en caché cualquier otra solicitud a medida que se realice, como se muestra en la siguiente lista.

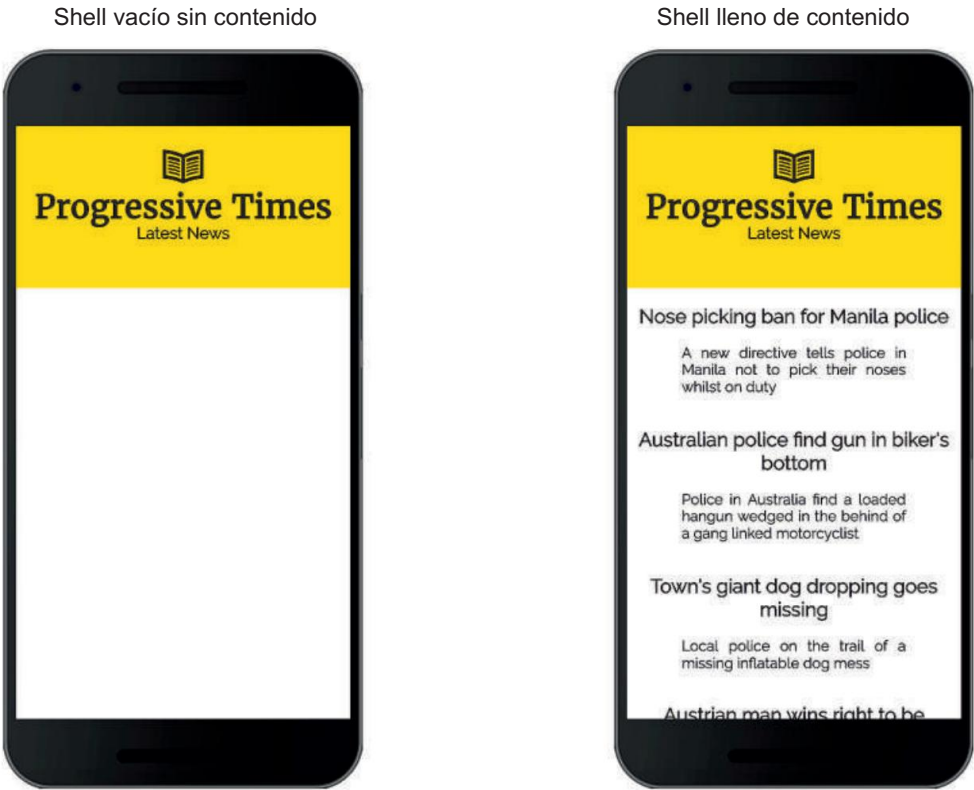


Figura 3.8 La aplicación de muestra Progressive Times utiliza la arquitectura App Shell.

Listado 3.6 Código de Service Worker para precaché y recursos Ccche durante el tiempo de ejecución

```
var cacheName = 'últimas noticias-v1';

// Almacenar en caché nuestros recursos conocidos durante la instalación
self.addEventListener('instalar', evento => {
  evento.esperar hasta (
    caches.open(nombrecaché)
      .entonces(caché => caché.addAll([
        './js/main.js',
        './js/articulo.js',
        './images/periódico.svg',
        './css/sitio.css',
        './data/latest.json',
        './data/data-1.json',
        './artículo.html',
        './index.html'
      ]))
  );
});
```

Abra el caché y almacene una variedad de recursos en el caché durante el tiempo de instalación.

```
// Almacenar en caché los recursos nuevos a medida que se obtienen
self.addEventListener('fetch', evento => { evento.respondWith(

    caches.match(event.request, {ignorarBuscar: verdadero}) .luego(función(respuesta)
    {
        si (respuesta) {
            respuesta de devolución;
        }
        var requestToCache = evento.request.clone();

        return fetch(requestToCache).luego( función(respuesta)
        {
            if(!respuesta || respuesta.status !== 200) {
                respuesta de devolución;
            }

            var respuestaToCache = respuesta.clone();
            caches.open(nombrecaché)
            .entonces(función(caché) {
                cache.put(requestToCache, respuestaToCache); });

            respuesta de devolución;
        });
    });
});
```

Diagram illustrating the logic flow of the code:

- Escuche el evento de recuperación.
- Ignore los parámetros de la cadena de consulta para no perder ningún caché.
- Si encontró una coincidencia exitosa, devuélvala en este punto y no continúe.
- Si no encontró nada en el caché, realice la solicitud
- Guárdelo en caché para que no tengamos que hacerlo. solicite de nuevo

El código del listado 3.6 es una combinación de almacenamiento en caché previo durante el tiempo de instalación y almacenamiento en caché a medida que recupera un recurso. La aplicación web utiliza una arquitectura App Shell, lo que significa que puede aprovechar el almacenamiento en caché de Service Worker para solicitar solo el datos necesarios para completar la página. Ya has almacenado correctamente los recursos para el shell, por lo que todo lo que queda es el contenido de noticias dinámico del servidor.

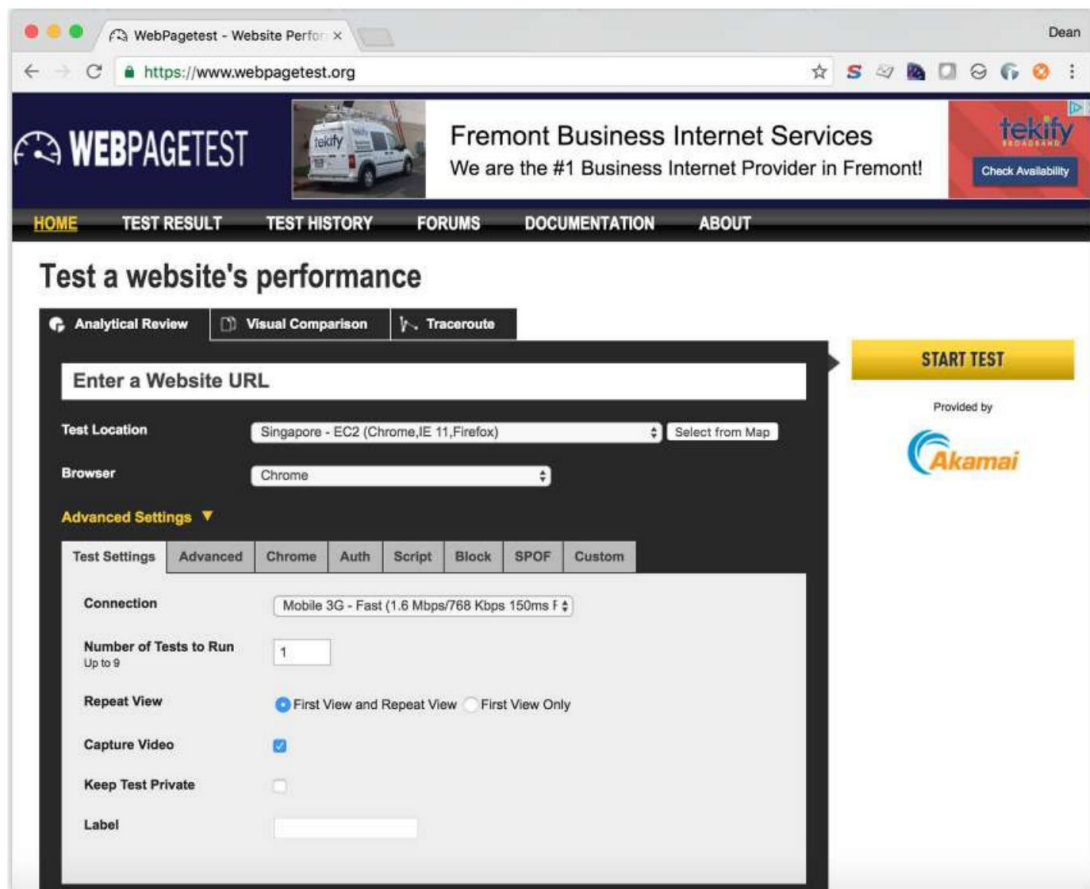
Si desea ver esta página web en acción, está disponible en GitHub y se puede acceder fácilmente a ella en bit.ly/chapter-pwa-3. De hecho, agregué todos los ejemplos de código que utilizar a lo largo de este libro para ese repositorio de GitHub.

Cada capítulo tiene un archivo Léame que explica lo que debe hacer para comenzar a construir y experimentar con el código de muestra en cada capítulo. Alrededor del 90% de los capítulos son código de interfaz de usuario, por lo que todo lo que necesita hacer es iniciar su host local y comenzar. Es También vale la pena señalar que debe ejecutar el código en <http://localhost> entorno y no en file:// entorno.

3.3 Comparación de rendimiento: antes y después del almacenamiento en caché

Llegados a este punto, espero haber logrado convencerles de lo fantástico que es el almacenamiento en caché de Service Worker. es. ¿¡Aún no!? Bien, bueno, con suerte las mejoras de rendimiento que obtendrás cuando El uso del almacenamiento en caché le hará cambiar de opinión.

Usando nuestra aplicación de muestra Progressive Times, podemos comparar la diferencia con y sin almacenamiento en caché de Service Worker. Una de mis formas favoritas de probar el mundo real. El rendimiento de un sitio web es utilizar una herramienta llamada [WebPagetest.org](https://www.webpagetest.org), se muestra en la figura 3.9.



The screenshot shows the WebPagetest website interface. The header includes the WebPagetest logo, a navigation menu with links like HOME, TEST RESULT, TEST HISTORY, FORUMS, DOCUMENTATION, and ABOUT, and a banner for Fremont Business Internet Services. The main content area is titled "Test a website's performance" and features three tabs: Analytical Review, Visual Comparison, and Traceroute. The Analytical Review tab is active, showing a form to "Enter a Website URL". Below this, there are dropdown menus for "Test Location" (set to Singapore - EC2) and "Browser" (set to Chrome). An "Advanced Settings" section is expanded, showing options for "Connection" (Mobile 3G - Fast), "Number of Tests to Run" (1), "Repeat View" (First View and Repeat View), "Capture Video" (checked), "Keep Test Private" (unchecked), and a "Label" field. A yellow "START TEST" button is visible on the right side of the form. The Akamai logo is also present, indicating the service provider.

Figura 3.9 [WebPagetest.org](https://www.webpagetest.org) es una herramienta gratuita que puedes utilizar para probar tus sitios web utilizando dispositivos reales de todo el mundo.

[WebPagetest.org](https://www.webpagetest.org) es una gran herramienta. Ingrese la URL de su sitio web y le permitirá crear un perfil de su sitio web desde cualquier ubicación del mundo utilizando un dispositivo del mundo real y una amplia gama de navegadores. Las pruebas se ejecutan en dispositivos reales y le brindan un desglose y un perfil útiles del rendimiento de su sitio web. Lo mejor de todo es que es de código abierto y de uso completamente gratuito.

Si ejecuto nuestra aplicación de muestra a través de [WebPagetest.org](https://www.webpagetest.org), produce algo similar a la figura 3.7.

Para probar el rendimiento de nuestra aplicación web de muestra en un dispositivo del mundo real, utilicé WebPagetest con una conexión móvil 2G desde un punto final en Singapur. Si alguna vez ha intentado acceder a un sitio web a través de una conexión de red lenta, sabrá lo molesto que puede ser esperar a que el sitio termine de cargarse. Como desarrolladores web, es importante que probemos nuestros sitios web tal como los usarían nuestros usuarios, y eso incluye

utilizando velocidades de conexión móviles más lentas y también dispositivos de gama baja. Una vez que WebPagetest completó el perfil de la aplicación web, produjo los resultados que se muestran en la figura 3.10.

	Load Time	First Byte	Start Render	Speed Index	Document Complete		
					Time	Requests	Bytes In
First View	12.258s	3.752s	7.481s	8030	12.258s	13	91 KB
Repeat View	0.578s	3.498s	0.476s	492	0.578s	0	0 KB

Figura 3.10 [WebPagetest.org](https://www.webpagetest.org) produce información útil sobre el rendimiento de su aplicación web mediante el uso de un dispositivo real.

En la primera vista, la página tardó unos 12 segundos en cargarse. Esto no es ideal, pero tampoco inesperado en una conexión 2G lenta. Pero si observa la vista repetida, el sitio se cargó en menos de 0,5 segundos y no realizó ninguna solicitud HTTP al servidor. La aplicación de muestra utilizó la arquitectura App Shell y, si recuerda el diseño, sabrá que cualquier solicitud futura se atenderá con la mayor rapidez porque los recursos necesarios ya se han almacenado en caché. Si se usa correctamente, el almacenamiento en caché de Service Worker mejora significativamente la velocidad general de su aplicación y mejora la experiencia de navegación independientemente del dispositivo o la conexión utilizada.

3.4 Profundizar en el almacenamiento en caché de Service Worker

En este capítulo, comenzamos a analizar cómo se puede utilizar el almacenamiento en caché de Service Worker para mejorar el rendimiento de su aplicación web. A medida que avancemos en el resto de este capítulo, veremos detenidamente cómo puede versionar sus archivos para garantizar que no haya discrepancias en la caché, así como para evitar algunos de los errores comunes que puede encontrar al usar Service Worker. almacenamiento en caché.

3.4.1 Versionando tus archivos

Habrà un momento en el que será necesario actualizar la caché de su Service Worker. Si realiza cambios en su aplicación web, asegúrese de que los usuarios reciban la versión más reciente de los archivos en lugar de las versiones anteriores. Como puede imaginar, entregar archivos antiguos por error causaría estragos en un sitio.

Lo mejor de Service Workers es que cada vez que realiza algún cambio en el archivo de Service Worker, se activa automáticamente el flujo de actualización de Service Worker.

En el capítulo 1, analizamos el ciclo de vida del trabajador de servicio. Recuerde que cuando un usuario navega a su sitio, el navegador intenta volver a descargar Service Worker en segundo plano. Si hay incluso un byte de diferencia en el archivo del Service Worker en comparación con lo que tiene actualmente, lo considera nuevo.

Esta útil funcionalidad le brinda la oportunidad perfecta para actualizar su caché con nuevos archivos. Puede utilizar dos enfoques al actualizar el caché. Primero, puede actualizar el nombre del caché que utiliza para almacenar. Refiriéndose al código

en el listado 3.2, puede ver la variable `cacheName` con un valor `'helloWorld'`. Si actualiza este valor a `'helloWorld-2'`, se creará automáticamente un nuevo caché y comenzará a servir sus archivos desde ese caché. La caché original quedaría huérfana y ya no se utilizaría.

La segunda opción, que personalmente creo que es la más segura, es versionar sus archivos. Esta técnica se conoce como eliminación de caché y existe desde hace muchos años. Cuando un archivo estático se almacena en caché, se puede almacenar durante largos períodos de tiempo antes de que caduque. Esto puede ser una molestia en caso de que realice una actualización en un sitio, pero debido a que la versión en caché del archivo se almacena en los navegadores de sus visitantes, es posible que no puedan ver los cambios realizados. La eliminación de caché resuelve este problema mediante el uso de un identificador único de versión de archivo para indicarle al navegador que hay una nueva versión del archivo disponible.

Por ejemplo, si agregara una referencia a un archivo JavaScript en HTML, Es posible que desee agregar una cadena hash al final del nombre del archivo, similar a esta:

```
<script type="text/javascript" src="/js/main-xtvbas65.js"></script>
```

La idea detrás de la eliminación de caché es crear un nombre de archivo completamente nuevo cada vez que realiza cambios en el archivo para garantizar que el navegador obtenga el contenido más actualizado posible. Imagine el siguiente escenario en nuestra aplicación web de periódico. Digamos que tiene un archivo llamado `main.js` y lo almacena en el caché exactamente como está. Dependiendo de cómo esté configurado su Service Worker, recuperará esta versión del archivo del caché cada vez. Si realiza un cambio en el archivo `main.js` con un código nuevo, Service Worker aún interceptará y devolverá la versión anterior en caché aunque desee servir la versión más nueva del archivo. Pero si cambia el nombre del archivo a, digamos, `main.v2.js` y actualiza su código para que apunte a esta nueva versión, puede asegurarse de que el navegador obtendrá la versión nueva cada vez. De esa manera, su periódico siempre entregará los resultados más recientes.

sus usuarios.

Existen muchos enfoques diferentes para implementar esta solución y todos ellos pueden depender de su entorno de codificación. Algunos desarrolladores prefieren generar estos nombres de archivos con hash durante el tiempo de compilación, y otros pueden hacerlo usando código y generar los nombres de archivos sobre la marcha. Cualquiera que sea el enfoque que utilice, esta técnica es una forma probada de garantizar que siempre entregue los archivos correctos.

3.4.2 Manejo de parámetros de consulta adicionales

Cuando un trabajador de servicio busca una respuesta almacenada en caché, utiliza una URL de solicitud como clave. De forma predeterminada, la URL de solicitud debe coincidir exactamente con la URL utilizada para almacenar la respuesta almacenada en caché, incluidos los parámetros de consulta en la parte de búsqueda de la URL.

Si realiza solicitudes HTTP para archivos adjuntos con cadenas de consulta que a veces cambian, esto podría terminar causándole algunos problemas. Por ejemplo, si realiza una solicitud de una URL que coincidió anteriormente, es posible que no aparezca porque la cadena de consulta difiere ligeramente. Para ignorar las cadenas de consulta cuando verifica el caché, use el atributo `ignoreSearch` y establezca el valor en verdadero, como se muestra en la siguiente lista.

Listado 3.7 Código de Service Worker para ignorar los parámetros de la cadena de consulta

```
self.addEventListener('buscar', función(evento) {  
  
    event.respondWith( caches.match(event.request,  
  
    { ignoreSearch: true })).luego(function(response) { return respuesta || fetch(event.request); }) ); });
```

El código del listado 3.7 utiliza la opción `ignoreSearch` para ignorar la parte de búsqueda de la URL tanto en el argumento de la solicitud como en las solicitudes almacenadas en caché. Puede ampliar esto aún más utilizando otras opciones de ignorar, como `ignoreMethod` e `ignoreVary`. Por ejemplo, el valor `ignoreMethod` ignorará el método del argumento de la solicitud, por lo que una solicitud POST puede coincidir con una entrada GET en la caché. El valor `ignoreVary` ignorará el encabezado de variación en las respuestas almacenadas en caché.

3.4.3 ¿Cuánta memoria necesitas?

Siempre que hablo con desarrolladores sobre el almacenamiento en caché de Service Worker, las preguntas que surgen regularmente tienen que ver con la memoria y el espacio de almacenamiento. ¿Cuánto espacio utiliza el Service Worker para almacenar en caché? ¿Cómo afectará este uso de memoria a mi dispositivo?

La respuesta honesta es que depende de su dispositivo y de las condiciones de almacenamiento. Como todo el almacenamiento del navegador, el navegador puede desecharlo si el dispositivo sufre presión de almacenamiento. Esto no es necesariamente un problema porque los datos se pueden recuperar de la red según sea necesario. En el capítulo 7, veremos otro tipo de almacenamiento llamado almacenamiento persistente que puede usarse para almacenar datos en caché de forma más permanente.

En este momento, los navegadores más antiguos todavía pueden almacenar respuestas en caché en su memoria y el espacio que usan no es diferente del espacio que usa el Service Worker para almacenar en caché los recursos. La única diferencia es que el almacenamiento en caché de Service Worker lo coloca a usted en el asiento del conductor y le permite crear, actualizar y eliminar entradas almacenadas en caché mediante programación, lo que le permite acceder a recursos sin una conexión de red.

3.4.4 Llevando el almacenamiento en caché al siguiente nivel: Workbox

Si escribe regularmente código en sus Service Workers que almacena recursos en caché, puede encontrar Workbox (<https://workboxjs.org/>) útil. Escrita por el equipo de Google, es una biblioteca de ayudas que le ayudarán a empezar a crear sus propios Service Workers en poco tiempo, con controladores integrados para cubrir las estrategias de red más comunes. En unas pocas líneas de código, puede decidir si desea servir recursos específicos únicamente desde la memoria caché, servir recursos desde la memoria caché y luego recurrir a ellos, o tal vez solo devolver recursos de la red y nunca almacenar en caché. Esta biblioteca le brinda control total sobre su estrategia de almacenamiento en caché. Ver figura 3.11.

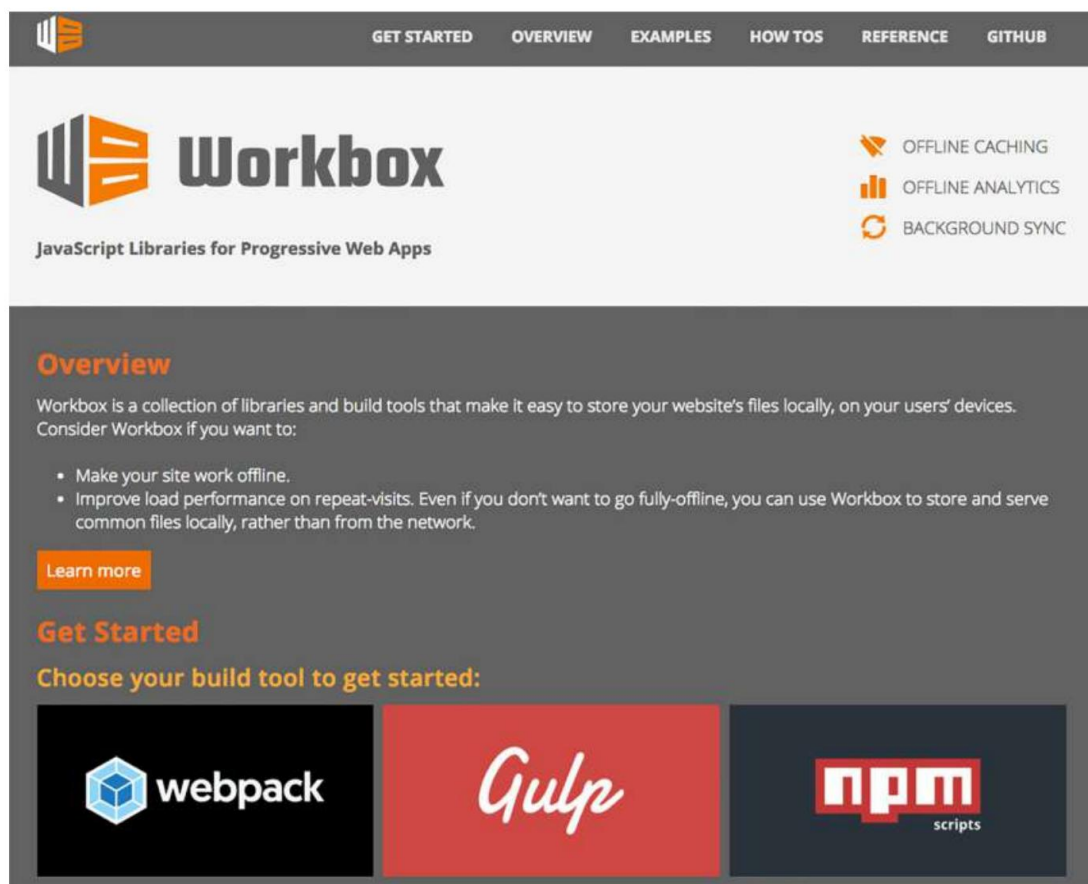


Figura 3.11 Workbox proporciona una biblioteca de ayudas para usar en la creación de sus propios trabajadores de servicio.

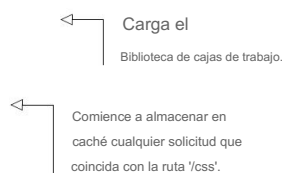
Workbox le proporciona una forma rápida y sencilla de reutilizar estrategias comunes de almacenamiento en caché de red en lugar de reescribirlas una y otra vez. Por ejemplo, supongamos que desea asegurarse de recuperar siempre sus archivos CSS del caché pero solo recurrir a la red si un recurso no está disponible. Al utilizar Workbox, usted registra a su trabajador de servicio de la misma manera que lo ha hecho a lo largo de este capítulo. Luego importa la biblioteca a su archivo Service Worker y comienza a definir las rutas que desea almacenar en caché.

En el listado 3.8, el código comienza importando la biblioteca Workbox usando la función `importScripts`. Los trabajadores de servicios tienen acceso a una función global, llamada `importScripts()`, que les permite importar scripts en el mismo dominio a su alcance. Esta es una forma práctica de cargar otro script en un script existente. Mantiene el código limpio y significa que solo carga el archivo cuando es necesario.

Listado 3.8 Usando Workbox

```
importScripts('workbox-sw.prod.v1.1.0.js');
const workboxSW = nuevo self.WorkboxSW();

caja de trabajoSW.router.registerRoute( 'https://
  test.org/css/(.*)',
  caja de trabajoSW.strategies.cacheFirst()
);
```



Una vez que se haya importado el script, puede comenzar a definir las rutas que desea almacenar en caché.

En el listado 3.8, estás definiendo una ruta para cualquier cosa que coincida con la ruta '/css/' y sirviéndolo siempre con un primer enfoque de caché. Esto significa que los recursos siempre se servirán desde la memoria caché y recurrirán a la red si no existen. Neceser de costura también proporciona una serie de otras estrategias de almacenamiento en caché integradas,² como solo caché, solo red, red primero, caché primero o más rápido, que intenta encontrar la respuesta más rápida desde la memoria caché o la red. Cada una de estas estrategias se puede aplicar a diferentes escenarios, e incluso puedes mezclarlos y combinarlos con diferentes rutas para lograr el mejor efecto.

Workbox también le proporciona funcionalidad para almacenar en caché los recursos. En el mismo forma en que almacenó en caché los recursos durante la instalación del Service Worker en el listado 3.2, puedes lograr esto con unas pocas líneas de código usando Workbox.

Cada vez que me acerco a un nuevo proyecto, sin duda mi biblioteca favorita para usar es Neceser de costura. Simplifica su código y le proporciona estrategias de almacenamiento en caché probadas y comprobadas que puede implementar en unas pocas líneas de código. De hecho, la PWA de Twitter que analizamos en el capítulo 2 utiliza Workbox para que el código sea más sencillo de entender y se basa en sobre estos enfoques de almacenamiento en caché probados y probados.

3.5 Resumen

El almacenamiento en caché HTTP es una manera fantástica de mejorar el rendimiento de su sitio web, pero no está exento de defectos.

El almacenamiento en caché de Service Worker es extremadamente poderoso porque le brinda programación control sobre exactamente cómo almacena en caché sus recursos. Cuando se usa mano a mano con Almacenamiento en caché HTTP, obtienes lo mejor de ambos mundos.

Si se utiliza correctamente, el almacenamiento en caché de Service Worker supone una enorme mejora del rendimiento y ahorro de ancho de banda.

Puede utilizar varios enfoques diferentes para almacenar en caché los recursos, y cada uno de ellos se pueden adaptar para satisfacer las necesidades de sus usuarios.

WebPagetest es una gran herramienta para probar el rendimiento de sus aplicaciones web utilizando dispositivos del mundo real.

Workbox es una biblioteca útil que le proporciona técnicas de almacenamiento en caché probadas y comprobadas.

² www.recode.net/2016/6/8/11883518/app-boom-over-snapchat-uber