Get started with Service Worker

tjoskar.github.io/service-worker-exercises/

37 stars 10 forks 4 watching Branches Tags Activity

Public repository

master 3 Branches 0 Tags Go to file t Go to file + Add file Code

Oskar Karlsson Fix quotation a8aeb3e · 7 years ago

images	Place vanilla example in the root	7 years ago
old-reference	Place vanilla example in the root	7 years ago
README.md	Fix quotation	7 years ago
icon.png	Place vanilla example in the root	7 years ago
index.html	Reset	7 years ago
index.js	Place vanilla example in the root	7 years ago
offline.gif	Place vanilla example in the root	7 years ago
style.css	Place vanilla example in the root	7 years ago

Let's get started

Prerequisite

You need a browser that supports service worker and async/await for these exercises. I recommend chrome but you should be able to use Opera and Firefox as well.

Hello friend

The first step is to clone this project:

```
$ git clone git@github.com:tjoskar/service-worker-exercises.git
```

Second, you need to serve the application on your computer. Service workers only run over HTTPS (for security reasons) but there is one exception and that is localhost (your true home). Start a local server and point out the `index.html` file. If you have python installed, you can just start a python server with: `python -m SimpleHTTPServer` or if you like you can install and use a separated lib: https://github.com/indexzero/http-server or maybe install an add-on to chrome: https://chrome.google.com/webstore/detail/web-server-for-chrome/ofhbbkphhbklhfoeikjpcbhemlocgigb

The possibilities are endless.

> Make sure that you serving the site from the root. Meaning: you should visit the site by going to something like `http://localhost:8000/` and not by any sub folder like: `http://localhost:8000/service-worker-exercises/`

.

> You will only need to make changes in your sw.js file. Except for the registration part, which you will do in index.html

Step 1

You should now be able to see the fantastic site, Happy News 🎉

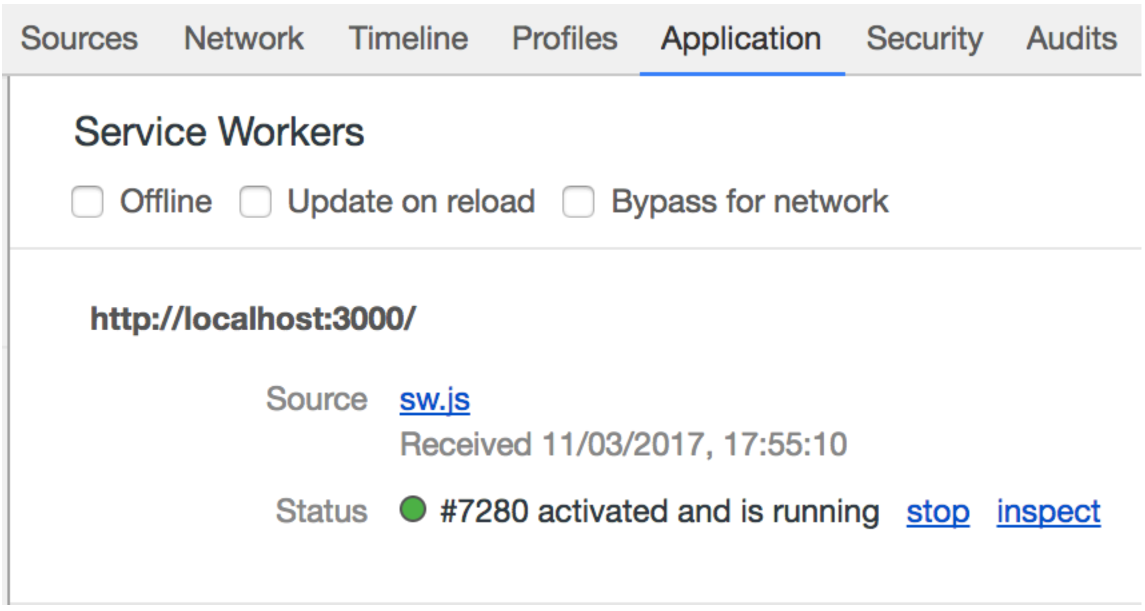
But it’s quite slow and if you break the Internet connection the site doesn’t work at all, that's a bummer.

Let's fix that problem with a service worker! First of, let's create a service worker by creating a new (empty) file that we call `sw.js` and place it beside `index.html`.

We now need to register the service worker. Open up `index.html` in your favorite editor and insert the following registration:

```
if ('serviceWorker' in window.navigator) {
  window.navigator.serviceWorker
    .register('/sw.js')
    .then(reg => {
        console.log('We are live 🚀', reg);
        return reg;
    })
    .catch(err => {
        console.log('ಠ_ಠ', err);
        throw err;
    });
} else {
  console.log('Your browser do not support Service Worker :/')
}
```

And we are live! If you are using Chrome you should be able to see the service worker under the Application tab.



http://localhost:3000/

Source sw.js
Received 11/03/2017, 17:55:10

Status #7280 activated and is running

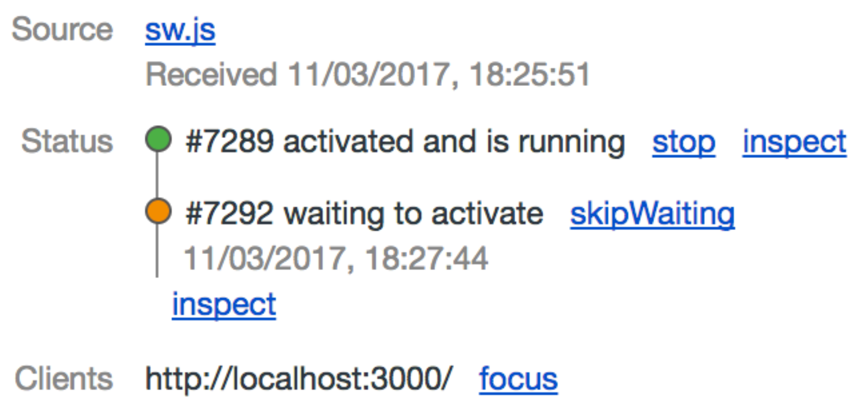
Okay, we have a service worker but it’s empty and doesn’t do anything :/ Let's fix that. Let’s start slow by trying to replace every image with a image of Nicolas cage.

Open `sw.js` in your editor and add an event-listener for `fetch` events:

```
self.addEventListener('fetch', event => {
  console.log('The user request: ', event.request.url);
  const url = new URL(event.request.url);

  // Check if the user requests an image
  if (/\.(png|jpg|jpeg|gif)/.test(url.pathname)) {
    // Hijack the fetch request and response with an image of Nicolas cage
    event.respondWith(fetch('http://i.imgur.com/xiJk0LHh.jpg'));
  }
});
```

Refresh your browser and.. nothing will change... Why is that? If you take a look under the Application tab, you will see that a new service worker is in the queue but the old one is still activated.


Source sw.js
Received 11/03/2017, 18:25:51
Status #7289 activated and is running stop inspect
#7292 waiting to activate skipWaiting
11/03/2017, 18:27:44
inspect
Clients http://localhost:3000/ focus

The browser will download the new service worker and install it but the old one will still be running (activated) until you close the tab/window (for UX reasons). This sucks if you develop a new service worker and wants to get the latest service worker every time your refresh the page.

Chrome comes to the rescue! If you check the box: `Update on reload` the browser will:

Download the service worker
Install it as a new version even if it's byte-identical
Skip the waiting phase so the new service worker activates immediately, meaning the browser will emit a new `install` and `activate` event for every refresh
Navigate the page

So, do yourself a favor and check the box (`Update on reload`).

Try once again to reload the page and you should see all images to be replaced with an image of Nicolas Cage.

Step 2

Okay, that's cute but let's make something useful.

Add a new event listener that listens for the install event and add all static files to the cache with the global `caches` -object:

```
const staticFiles = [
    '/',
    '/index.js',
    '/style.css',
    'https://fonts.googleapis.com/css?family=Open+Sans'
];
const saveCacheFiles = caches.open('you-cache-storage-name').then(cache => cache.addAll(staticFiles));
```

You will need to use `event.waitUntil()` to extends the lifetime of the event and prevent the browser from terminating the service worker before asynchronous operations within the event have completed.

Don't know how to do it? Take a look at one possible solution here.

Nice, we have now pre fetched all static files but we don't use them, yet. Let's update our callback for the `fetch` -handler to check for request for static files and grab them from the cache:

```
self.addEventListener('fetch', event => {
  console.log('The user request: ', event.request.url);
  const url = new URL(event.request.url);

  if (location.hostname === url.hostname) {
    event.respondWith(caches.open('you-cache-storage-name').then(cache => cache.match(event.request)));
  } else if (/\.(png|jpg|jpeg|gif)/.test(url.pathname)) {
    event.respondWith(fetch('http://i.imgur.com/xiJk0LHh.jpg'));
  }
});
```

You should now be able to get the html, css and js files even though you are offline or on Lie-Fi.

Bonus: The font from google is still out of reach though. You will need to update the predicate to something like: `[location.hostname, 'fonts.googleapis.com', 'fonts.gstatic.com'].includes(url.hostname)` and add the files to the cache to get it work.

Step 3

If you now uncheck the box for "Update on reload" and reload the page you may see that we have a problem. The browser will never re-fetch the static files again 😱 We will never download new versions of e.g. index.html. That's because we only lookup the files from the cache and never do a fetch. The browser will however reload the cache if you make a change to the server worker (sw.js). Why?

Let's fix this problem. Let's create a function that response with a value from cache and then tries to fetch a new value from the server and update the cache if it succeeds.

Such function could look something like this:

```
async function cacheFallbackOnNetwork(request, cacheName) {
    const cacheResponse = await fromCache(request, cacheName);
    const fetchPromise = updateCache(request, cacheName);
    return cacheResponse || fetchPromise;
}

async function fromCache(request, cacheName) {
    const cache = await caches.open(cacheName);
    return cache.match(request);
}

async function updateCache(request, cacheName) {
    const cache = await caches.open(cacheName);
    const response = await fetch(request);
    if (response.ok || response.type === 'opaque') {
        await cache.put(request, response.clone());
    }
    return response;
}
```

Nice, we should now update our `fetch` -eventhandler to use `cacheFallbackOnNetwork` instead of just going for the cache.

You should now see under the network tab that the service worker make new request for the static files 🎉

Step 4

It should be nice to get the news from `https://happy-news-nmnepmqeqo.now.sh/` even though you are on offline or on Lie-Fi.

Create a function that tries to fetch the resource from Internet and fall back on cache on failure.

No idea how to do it? Take a look at this proposed solution.

Step 5

We will at this point get the news if we are offline but the images are still absent. Let's fix that with something like this in our fetch event handler.

```
if (url.hostname === 'i.redditmedia.com') {
    event.respondWith(cacheFallbackOnNetwork(event.request, imageCacheName));
}
```

Step 6

All right, we can now get both the news list and images when we are offline but the user experience is still quite bad on Lie-Fi. The user still needs to wait for the news list before the user can interact with the application. That is not good. – One solution could be to fetch the news list from the cache, which would decrease the waiting time to almost zero! But... that means the user only get old news and not the latest :/

What if we could load the news list from cache for fast responses and then update the cache entry from the network. When the network response is ready, the UI updates automatically. – That sound like a good plan. Let's try it out.

First we need to grab the response from the cache. We can do that with `fromCache` (which we created in step 3):

```
const cacheResponse = fromCache(event.request, cacheName);
```

We also need to grab an updated version of the resource from the server and update the cache. We can use `updateCache` (which we also created in step 3) for that purpose:

```
const updatedResponse = updateCache(event.request, cacheName);
```

If we found a match in the cache we should respond to the user with that version but if we do not found any match we should return the fetched value. We can achieve this by:

```
const cacheOrNetwork = cacheResponse.then(response => {
    // This function is called when we got a result from the cache lookup
    if (response === undefined) {
        // If the response is undefined we don't have this request in the cache and we should give the user the result from the
        return updatedResponse;
    } else {
        // All right, we have a cached value. Give it to the user
        return response;
    }
});
event.respondWith(cacheOrNetwork); // Take over the fetch request and response with either a cached result or a response from tl
```

We can shorten the code above with the following:

```
const cacheOrNetwork = cacheResponse.then(response => response || updatedResponse);
event.respondWith(cacheOrNetwork);
```

If we response with the fetch request we are done, the user gets the latest content and we do not have to do anything more. But if we response with a cached value we should inform the user when we have an updated value. We can use the following code to achieve this behavior:

```
const refreshResponse = cacheResponse
    .then(response => {
        if (response === undefined) {
            // We did found response in the cache, meaning the user get some old value so let's switch over to the fetch request
            return updatedResponse;
        }
    })
    .then(response => {
        if (response) {
            // `response` is the fetch response here
            // We should inform the user somehow here.
        }
    });
event.waitUntil(refreshResponse);
```

We could use `client.postMessage` to inform the user about the update:

```
async function postMessage(response, type) {
    const message = JSON.stringify({ url: response.url, type });
    const clients = await self.clients.matchAll();
    clients.forEach(client => client.postMessage(message));
}
```

So the overall code should look something like:

```
function cacheAndUpdate(event, cacheName) {
    const cacheResponse = fromCache(event.request, cacheName);
    const updatedResponse = updateCache(event.request, cacheName);
    const cacheOrNetwork = cacheResponse.then(response => response || updatedResponse);
    const refreshResponse = cacheResponse
        .then(response => {
            if (response) {
                return updatedResponse;
            }
        })
        .then(response => {
            if (response) {
                return postMessage(response, 'refresh-news-list');
            }
        });
    event.respondWith(cacheOrNetwork);
    event.waitUntil(refreshResponse);
}

async function postMessage(response, type) {
    const message = JSON.stringify({ url: response.url, type });
    const clients = await self.clients.matchAll();
    clients.forEach(client => client.postMessage(message));
}
```

We can now use `cacheAndUpdate` in our fetch-event handler to handle api calls:

```
if(url.hostname === 'happy-news-hcooumiahc.now.sh') {
    cacheAndUpdate(event, apiCacheName);
}
```

Reload the webpage and you should see a nice-looking popup that ask you if you want to refresh the webpage. Take a look at `messageHandler` in `index.js`, do you get what is happening? If not, ask your neighbor or ask me. Please do. I'm quite lonely.

Step 7

Let's solve another kind of problem. Close your eyes, turn off your mind, relax and float down stream. Lay down all thoughts, surrender to the void. You are on a train and you want to read some happy news. You open up the webpage, which loads instant thanks to service worker and your incredible programing skills. You click on an article that interests you. Your phone says that you have a good WiFi connection but it's a lie. You are on Lie-Fi. The spinner just keeps on going and you start to get frustrated. You want to read happy news but all you get is this spinner that keeps on forever. WHAT THE FUDGE? It's 2017 and you must watch this stupid white screen on this dumb phone on this dumb train. The person next to you accidentally spill coffee on your shoes and you snap and kills him with your cellphone. You spend the rest of your life in prison. **snap** you are awake again. This is a real problem. Do not let loading screens kill people.

How can we fix this? We can't cache the whole world and we do not know what article the user wants to read in advance so we can't prefetch the article. But what we can do is to inform the user when the article is available so the user don't have to look at a white screen while waiting. Meaning:

The user clicks on an article

If it takes more then 3s to load the article, tell the user that he can close the browser and that he/she will get a push notification when we have the article in the cache

The service worker downloads the article in the background

When we have it, we send a push notification and inform the user that one can read the article and no one needs to get injured.

We can use the sync event for this:

```
self.addEventListener('sync', event => {
    console.log('sync event: ', event);
});
```

Add the code above in your service worker, reload the page and make sure that we have the latest service worker. Throttle the network speed and try to read an article. You should now get a console log from the service worker. Do you?

> We will misuse the tag in this exercise to keep down the complexity. You should use a queue database like indexedDB instead of relying on the tag string.

Try to fetch the requested article, update the cache and send a notification. Good to know: `event.tag` will be `news-article-{id}`, eg. `news-article-13` or `news-article-666` i.e. you can get the id with somthing like:

```
self.addEventListener('sync', event => {
    console.log('sync event: ', event);
    if (event.tag.startsWith('news-article-')) {
        const articleId = Number(event.tag.substr(13));
    }
});
```

You can show a notification with the following api:

```
await registration.showNotification('The article is ready for you', {
    icon: 'icon.png',
    body: 'View the article',
```

I leave the rest to you but if you get stuck you can take a look at this code.

Okay, you should now get a notification when the article is ready (remember that the sync event is quite irregular so you do not know when it runs).

We now need to tell the service worker what to do when the user clicks on the notification:

```
self.addEventListener('notificationclick', event => {
    // Assuming only one type of notification right now
    event.notification.close();
    clients.openWindow(`${location.origin}/#news/${event.notification.data}`);
});
```

Step 8

It should be nice if we could push some notification from a backend server.

With service worker we can.

First we need to listens for push events.

```
self.addEventListener('push', event => {
    console.log(`Push event: "${event.data.text()}"`);

    const title = 'New happy news!';
    const options = {
        body: 'Nice!',
        icon: 'icon.png',
        data: event.data.text()
    };

    event.waitUntil(self.registration.showNotification(title, options));
});
```

You can try out this code by clicking on `push` under the application tab in chrome.

It should also be nice to send a push notification, even though we don't have the tab opened.

To be able to send a notification we need to get the push subscription: you can use this function: `getPushSubscription()` in the console to get the all the data you need.

Make the following request from ex. Postman/Insomnia/curl:

```
POST https://happy-news-nmnepmqeqo.now.sh/notify
{
    "endpoint": "your-endpoint",
    "keys": {
        "p256dh": "your-key",
        "auth": "your-auth-key"
    },
    "articleId": "articleId"
}
```

And you should get a notification 🎉

Step 9

Compare your sw.js file with mine. Did you do anything different? Tell me!

Step 10

Releases

No releases published

README

Deployments 5

✓ **github-pages** 7 years ago

+ 4 deployments

Languages

● **JavaScript** 96.2% ● **TypeScript** 2.8% ● **Other** 1.0%