01

# Nossa Web App offline com Service Workers e Cache Storage

### Transcrição

No último vídeo, acessamos as imagens via Cache Storage, em vez do servidor. Utilizamos o Service Worker para interceptar todos os pedidos feitos pela página. Quando começamos a usar um Service Worker, acabamos deixando de usar o Application Cache, e tanto o Service Worker quanto o Cache Storage vieram para substitui-lo.

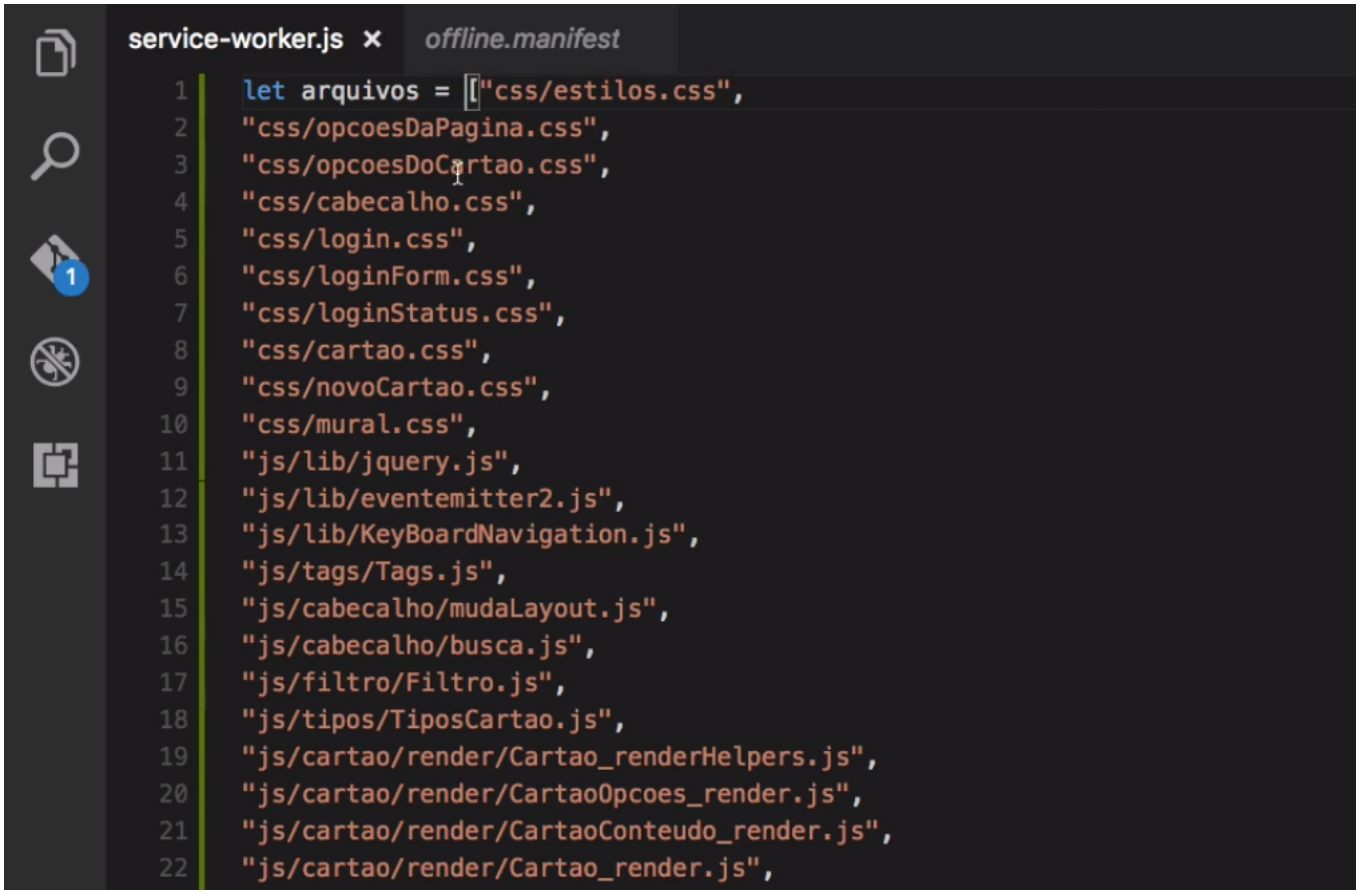
Simulei estar offline quando supostamente as imagens deveriam vir do Application Cache. Se recarrego a página, ainda no modo offline, não tenho mais Application Cache, então temos que utilizar o Service Worker e o Cache Storage para armazenamento dos arquivos.

O Service Worker é muito poderoso, e até então usamos apenas um pequeno pedaço deste poder. Ouvimos o evento de `fetch()`, cujo código é executado sempre que nosso site solicitar um arquivo. Isto, no entanto, é só uma das coisas que o site pode fazer, algo que ele já faz normalmente com as tags `<script>`, de links, em CSS e JAVASCRIPT.

Vou parar de usar o termo "site" e voltar a falar que nossa página é mais do que isso, ele não fica apenas carregando vários arquivos, é um aplicativo. E o que aplicativos têm em comum entre si? São instalados. E é disso que faltou falarmos em relação ao Service Worker.

O que significa instalar? Tem a ver com algo muito similar ao que fazíamos no Application Cache, em que listávamos todos os arquivos a serem armazenados no `offline.manifest`. O que acontece agora é que esta mesma lista precisa ser repassada ao nosso cache. Ou seja, o Application Cache deixa de existir, porém o Cache Storage ainda consegue armazenar arquivos.

Para salvarmos os arquivos desejados no Cache Storage, vamos transformá-los numa lista de JAVASCRIPT, dentro da qual armazenarei os nomes dos arquivos. O que eles têm em comum, nesse momento, é que são textos. Por motivos de praticidade, vou usar a ferramenta de seleção múltipla colocando aspas antes e depois de uma só vez, tendo assim uma lista com os nomes de todos os arquivos a serem salvos no Cache Storage:




Para armazenar arquivos no Cache Storage preciso abrir um cache (através de `caches.open`), porém, a única fonte que tinha até agora é `"ceep-imagens"`. Como não estou mais lidando com imagens, trocarei este nome para `"ceep-arquivos"` (dentro do arquivo denominado `service-worler.js`). Criada a lista de arquivos, é necessário navegar nela, sendo que para cada arquivo preciso colocar uma resposta dentro do cache, pois isso é que é armazenado. Para cada pedido, haverá uma resposta cacheada. Para cada um dos nomes da lista, tenho que pedir acesso à respectiva resposta ao servidor, para então serem armazenados... Daria um trabalho enorme!

O Cache Storage nos auxilia nisto pois foi pensado juntamente com os Service Workers para instalação da app e possibilidade de adicionar vários arquivos nela. Para tal, cabe a função `addAll`, para o qual passarei uma lista com todas as urls e nomes dos arquivos que quero cachear. Os pedidos serão feitos ao servidor, cujas respostas serão cacheadas automaticamente. Este é o código que precisaremos implementar no arquivo `service-worker.js`:

```
caches.open("ceep-arquivos").then(cache => {
  cache.addAll(arquivos)
  })
```

Esse código criado, no entanto, está "solto" dentro do `service-worker.js`. Quando é que ele será executado? Em um teste inicial que tínhamos feito com `console.log`, mostra-se que todo código "solto" é executado quando o Service Worker é registrado (no arquivo `registra.js`), o qual é executado a partir da tag `script` em `index.html`. Sempre que a página é recarregada, sua execução é realizada.

Ou seja, invariavelmente, quando acessar a página estarei adicionando os arquivos no cache novamente, solicitando ao servidor e armazenando estes pedidos no Cache Storage. Caso não queiramos fazer isto repetidas vezes, existe a possibilidade de fazê-lo apenas quando o Service Worker for atualizado, ou instalado.

Além de ser registrado, o Service Worker é instalado e, posteriormente, será avaliado para ativação. Já passamos por isso em outras vezes. Para que o código seja rodado quando o Service Worker for instalado, repetiremos as ações de quando

definimos que a função `event` deve ser executada no `fetch()`. Há também um evento que posso ouvir, o de `install`:

```
self.addEventListener("install", function(){
  caches.open("ceep-arquivos").then(cache => {
    cache.addAll(arquivos)
    })
  })
```

Este código só será acessado uma vez, quando o Service Worker for instalado. Com o evento `install`, acessa-se o cache e todos os arquivos são salvos, os quais por enquanto são todos `.css` e `.js`, provenientes do arquivo `offline.manifest`.

Existe um arquivo muito importante que não foi acrescentado a esta lista. No Application Cache não precisávamos falar disto, mas o `index.html` é um arquivo que precisa estar disponível offline, pois trata-se do arquivo principal.

Nos Service Workers, não informamos que o endereço principal da página precisa ser cacheado. Acrescentamos, então, `"/"` à lista de arquivos em `service-worker.js`, remetendo à url de que será feito o pedido, carregado quando acesso o localhost (http://localhost:3000) a partir do servidor. Digo a ele que cacheie a url principal, o `"/"`.

Antes de recarregarmos a página, há algumas modificações a serem feitas: agora que estamos instalando a aplicação e salvando os arquivos no Cache Storage, também precisaremos ensinar ao Service Worker que ele deve procurar os arquivos no `ceep-arquivos`. No momento, nosso evento de `fetch()` só procura aqueles encontrados dentro de `ceep-imagens`. Teríamos que criar uma maneira de procurá-los nestes dois locais, e a decisão será feita com base na origem da resposta correspondente.

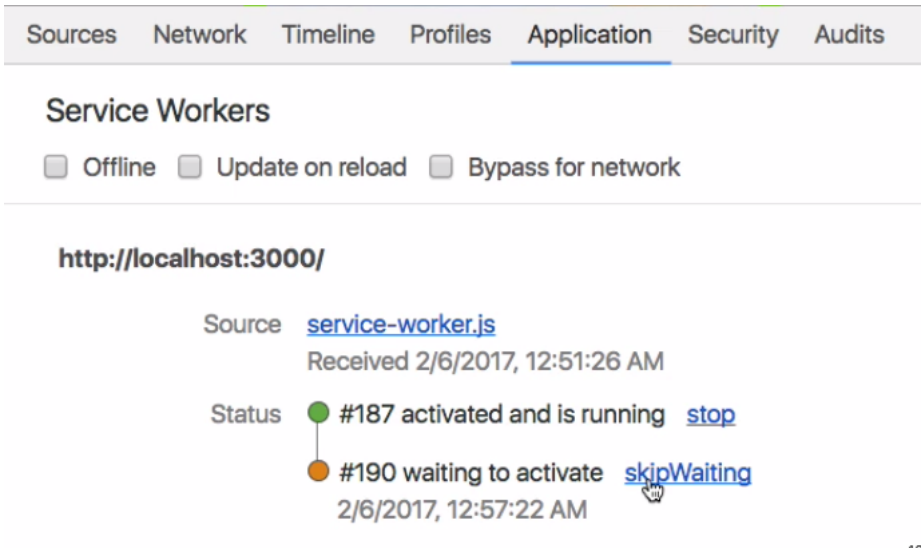
Este código não é muito trivial, e o Cache Storage também já está preparado para isso. Para procurar um arquivo em mais de um cache, não precisamos necessariamente procurá-lo dentro de um específico, e sim em todos eles. O que acontecia antes era abrirmos o cache ( `caches.open` ), utilizando depois a `cache.match`. Agora, em vez de escrever todo esse código, vou utilizar o `caches.match`, em que passo o pedido ou request daquilo que quero que seja procurado no cache.

```
self.addEventListener("install", function(){
  console.log("Instalou")
  caches.open("ceep-arquivos").then(cache => {
    cache.addAll(arquivos)
    })
  })

self.addEventListener("fetch", function(event){

  let pedido = event.request
  let promiseResposta = caches.match(pedido).then(respostaCache => {
    let resposta = respostaCache ? respostaCache : fetch(pedido)
    return resposta
    })

    event.respondWith(promiseResposta)

  })
```

Com isto estarei procurando o arquivo em todos os caches, não importa qual. Se houver alguma resposta, é ela que será usada. Vamos testar isto indo ao navegador e verificando o processo de instalação do Service Worker novo, com o Cache Storage acessando os novos arquivos. Recarregarei a página e irei à aba "Application", na parte do "Cache Storage",

confirmando a criação de `ceep-arquivos` e que todos os arquivos solicitados foram cacheados. O processo de instalação do novo Service Worker funcionou perfeitamente, embora não esteja sendo utilizado.

O Service Worker antigo é que está sendo usado, o que significa que existem dois deles ali, ambos instalados, apenas um ativo. Para fazer isto com o mais recente, clicarei em "*skipWaiting*" ao lado do nome do segundo Service Worker, como já visto anteriormente:




A maneira usual encontrada pelo usuário para ativação automática do Service Worker novo é fechar a aba e reabrí-la. Quando todas as abas da aplicação estiverem fechadas, a atualização é feita. Se quisermos fazer isto manualmente, é só clicar em "skipWaiting".

Pronto, agora o **Event Listener** é o novo, que pega os arquivos de todos os caches. Vamos testá-lo para verificar se isto é verdadeiro quando estamos offline, recarregando a página. Quando offline, todos os arquivos vieram do Service Worker, e a app continua funcionando.