04

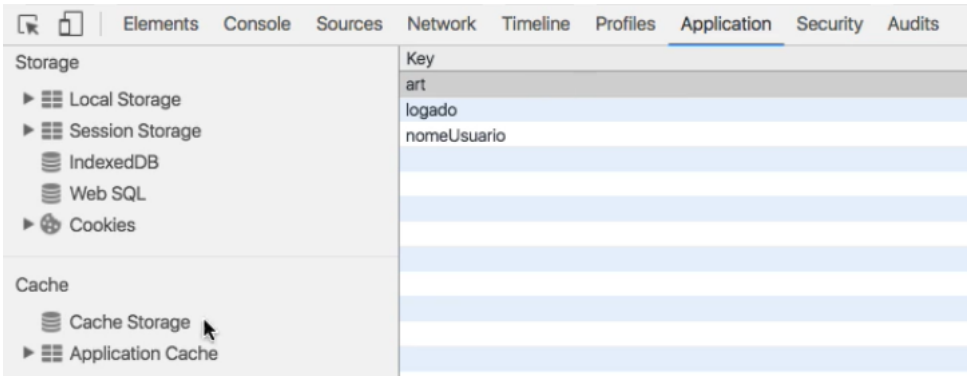
Salvando conteúdo dinâmico com Cache Storage

Transcrição

No vídeo anterior vimos que o Application Cache apresenta um problema bem chato: as imagens dos cartões cujas urls colocamos nessa sintaxe específica precisava funcionar offline. Significa que precisávamos salvar o que carregávamos dessa url da imagem, e para salvar esse arquivo externo, estávamos armazenando tais informações no Application Cache. Tudo que ele lista vem daquele arquivo intitulado `offline.manifest`, o qual nós mesmos criamos. Teríamos que fazer um código lá no servidor para fornecer um arquivo diferente para cada usuário, isto daria um enorme trabalho!

A W3C (*World Wide Web Consortium*) sabia que o Application Cache dificulta aplicações deste tipo, com conteúdo "cacheado" (de cache) dinâmico, que varia de usuário para usuário. A solução encontrada pela W3C foi a criação de uma nova especificação. Veremos agora uma forma alternativa para cachear arquivos externos. Deixaremos de ter um arquivo `offline.manifest`, o qual lista todos os arquivos necessários. Na verdade, o que faremos a partir de agora é acessar um lugar onde conseguimos colocar o que quisermos. Já fizemos isto para armazenar os textos dos cartões através do `localStorage`. A diferença é que, neste momento, queremos acessar um local para colocarmos qualquer tipo de arquivo vindo de fonte externa, não somente textos.

Então, não iremos mais acessar o `localStorage`, e sim o **Cache Storage**. Ele já possui até um ícone na aba "Application" e se encontra no mesmo lugar que o Application Cache:

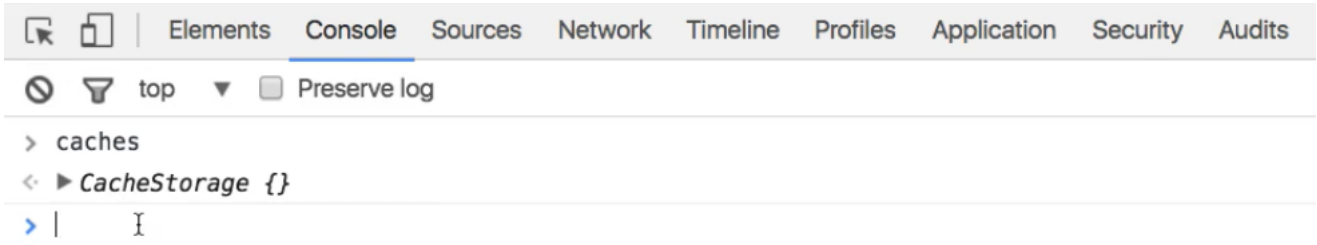


A ideia é que o Cache Storage, uma API (*Application Programming Interface*, ou Interface de Programação de Aplicações) seja o novo sucessor das funções atribuídas ao Application Cache, sendo este um tanto menos complexo. No Cache Storage, o que será armazenado são os arquivos externos de que o usuário precisa.

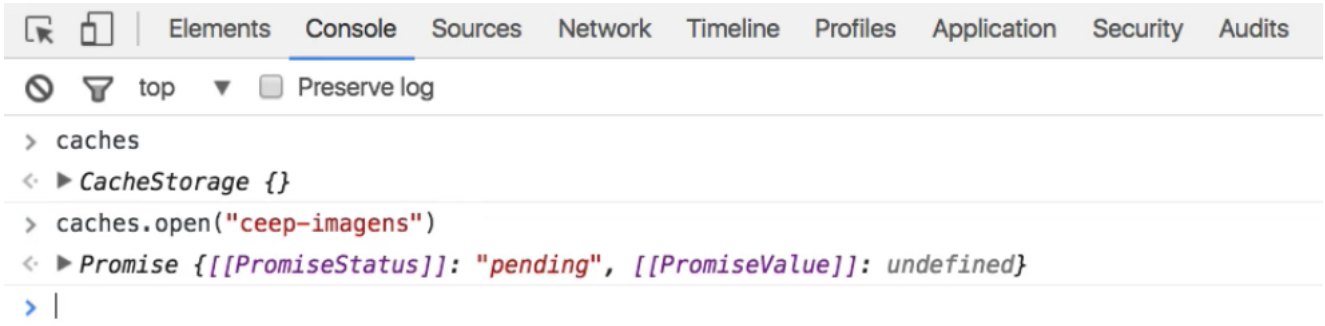
O que precisamos fazer para que o Cache Storage entenda isso, que queremos salvar a imagem do cachorro fofo, por exemplo?

Até agora, o que foi armazenado no `localStorage` foi a url da imagem escolhida pelo usuário. Dentro do cartão do usuário, portanto, temos a url, ou seja, já temos o endereço desta imagem. Se o Local Storage armazena uma propriedade com o nome do usuário, e um valor ("art", por exemplo), o que o Cache Storage armazenará, em contrapartida, é uma url que abriga um valor. Este valor é a imagem do cachorro fofo, neste exemplo. Aprenderemos a mexer no Cache Storage agora.

Testaremos algumas coisas aqui no console. Para termos acesso ao Cache Storage, o que temos no navegador é uma variável chamada `caches`. Darei um "Enter", ao qual ele me retorna:



A variável `caches` não está no plural à toa, existe um bom motivo para isto. No Cache Storage, não salvaremos vários arquivos simplesmente. Armazenaremos um pouco mais do que isto, vários caches distintos. Podemos ter, então, caches somente para imagens, outro para arquivos `.css`, podemos separá-los... Ou seja, `caches` armazena um ou vários caches, e quando estamos mexendo no Caches Storage, conseguimos abrir um cache chamado `ceep-imagens`, que é o que faremos agora, seguido de um "Enter":



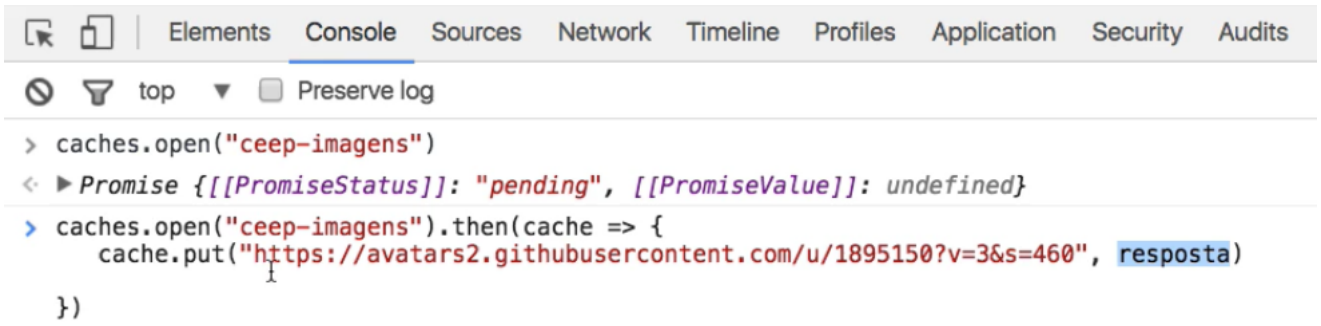
Vamos ver o que aconteceu? Não tem nada lá no Local Storage, que estranho. O que acontece é que precisamos atualizar a página para verificar estas modificações. Fazendo isto, pode-se ver que algo novo aparece: temos um cache chamado `ceep-imagens`. Não há nada dentro dele ainda, pois ele foi apenas criado, ficando disponível para armazenamento de arquivos. As colunas "#", "Request" e "Response" nos trarão informações úteis, de pedidos e respostas. Mais especificamente, pedimos um arquivo a partir de uma url, e teremos uma resposta proveniente desta solicitação.

O arquivo salvo em um cache em si é sempre uma url e a resposta dela.

Clicaremos na aba "Console", então, e preciso acessar o `ceep-imagens` para colocar nele um arquivo. Isto é possível de ser feito da mesma forma que o criamos. O `caches.open` criará um cache se o mesmo não existir ainda e, caso contrário, ele apenas o abrirá. Quando isto ocorrer, preciso conseguir acessá-lo. Darei um "Enter" para demonstrar que, o que o `caches.open` me retorna neste caso é uma `Promise`, como visto anteriormente.

A `Promise` nos diz que abrir um cache é algo síncrono, quero dizer, tenho acesso ao cache somente quando a `Promise` for resolvida, quando ela executará a função que eu passar a ela, a `then()`. Tenho, portanto, uma função de callback, o qual permitirá acesso ao cache como parâmetro, sendo possível adicionar arquivos dentro dele. Mais especificamente, pedidos ("Requests") e respostas ("Responses"). O `cache.put()` é uma função que executaremos colocando a url que queremos salvar, e sua resposta. Se já temos a url ou não, depende da imagem.

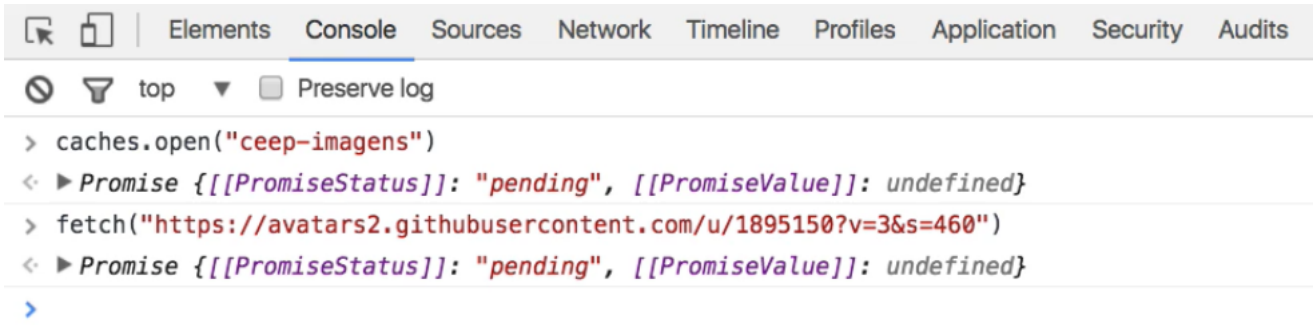
Por exemplo, quero utilizar minha imagem de perfil do GitHub, por exemplo, copio a url dela (https://avatars2.githubusercontent.com/u/1895150?v=3&s=460) e colo no código para salvá-la no cache.



O que coloco na resposta? Como pego um arquivo do servidor e coloco no cache? É um pouco diferente do que fizemos até então. Não tenho esse valor ainda, apenas a url, cuja imagem correspondente precisa ser solicitada. Para carregarmos essa imagem, precisamos executar uma função chamada `fetch()`:

```
fetch("https://avatars2.githubusercontent.com/u/1895150?v=3&s=460")
```

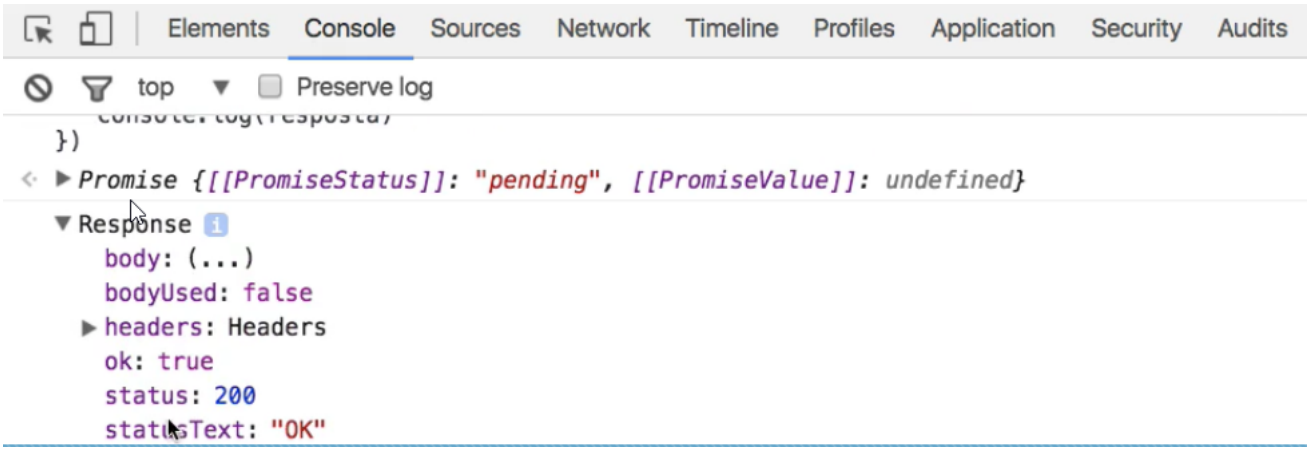
Apertamos o "Enter" para verificar o que acontece: temos como retorno mais uma `Promise`:



Significa que carregar um arquivo é algo que pode demorar, ou seja, é assíncrono também. Quando a imagem for carregada, a função `then()`, de callback, será executada. Esta será nossa resposta. Consigo inclusive chamar um `console.log` para darmos uma olhada em seu conteúdo:

```
fetch("https://avatars2.githubusercontent.com/u/1895150?v=3&s=460").then(resposta => {
  console.log(resposta)
  })
```

A última linha será portanto uma resposta, a qual me mostra a url, `body`, cabeçalhos (`headers`), entre outras informações, numa resposta completa do HTTP.

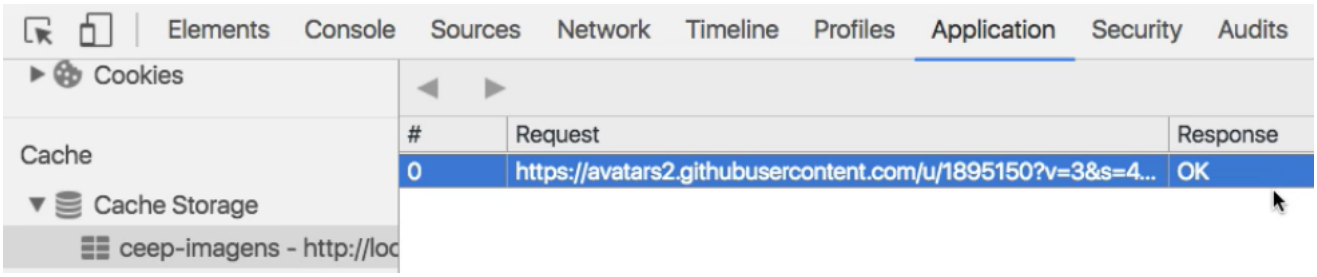


E é exatamente esta a resposta que armazenaremos lá no Cache Storage. Precisamos então solicitar a informação, através do `fetch`, para obter uma resposta, cujo acesso se dá pela função de callback, dentro da qual faremos o que já fizemos anteriormente: abriremos o cache chamado `ceep-imagens` e, quando ele for aberto, serei informado, dando-me um cache através da função de callback. Agora que posso acessar o cache, consigo colocar dentro dele a nossa url e resposta.

```
fetch("https://avatars2.githubusercontent.com/u/1895150?v=3&s=460").then(resposta => {
  caches.open("ceep-imagens").then(cache => {
    cache.put("https://avatars2.githubusercontent.com/u/1895150?v=3&s=460", resposta)
    })
  })
```

A resposta será a mesma que foi retornada ao `fetch`, e agora posso abrir um `ceep-imagens` pelo `cache.open` e, quando isto ocorrer, colocar dentro do Cache Storage. Para a url da imagem, quero colocar a resposta recebida pelo `fetch`. Darei um "Enter" e temos uma `Promise`.

Colocar informações no Cache Storage continua sendo síncrono. Se eu for à aba "Application", não tenho nada. Recarrego a página e clico em "Cache Storage > ceep-imagens", a url aparece ali, e tenho a resposta, "OK", que indica que deu certo:



Ou seja, agora tenho acesso à imagem mesmo quando estiver offline. Porém, não é qualquer imagem que queremos colocar lá no Cache Storage. Precisamos de uma maneira para pegar qualquer imagem ou url que o usuário colocar em seus cartões. É isso que veremos no próximo vídeo. Até lá!