01

Usando as imagens do Cache Storage

Transcrição

No último vídeo vimos como armazenar as informações dinâmicas no Cache Storage. Na aba "Application", consigo ver isto acontecendo: tenho um cache chamado `ceep-imagens`, e a url dentro dele. O funcionamento do Cache Storage se dá a partir do armazenamento da imagem e, caso queiramos utilizar esta informação, precisamos acrescentar o código correspondente.

Quando estou online, o conteúdo do cartão é uma url. A imagem possui um `src` (source) que é buscado no servidor, na rede. É assim que as imagens aparecem atualmente. O texto do cartão é convertido em imagem, por meio de uma tag `img`. Precisamos colocar a imagem no mural, com sua url vindo do Cache Storage e não do servidor.

Vamos ao código do `Mural.js` para encontrar o local por onde todos os cartões passam. No caso, todos eles passam pela função `adiciona(cartao)`, como discutido anteriormente. Precisamos da imagem antes do mural ser renderizado (através de `render()`), portanto:

```
function adiciona(cartao){
  if(logado){
    cartoes.push(cartao)
    salvaCartoes()
    cartao.on("mudanca.**", render)
    preparaCartao(cartao)
    let listaImagens = Cartao.pegaImagens(cartao)
    render()
    return true
  } else {
    alert("Você não está logado")
  }
}
```

A função `Cartao.pegaImagens(cartao)` lista todas as imagens do cartão. Para acessá-las no cache, uma vez que tenho essa lista em mãos, preciso inicialmente utilizar a `caches.open("ceep-imagens")`. Quando isto ocorrer, preciso voltar ao cache, usando a função `then()`. Para buscar cada imagem listada, utilizo `cache.match()` que, sozinha, recebe apenas a url. Quero que ela busque todas as imagens, portanto substituo-a por `cache.matchAll()` e, depois, terei acesso a todas as imagens.

```
function adiciona(cartao){
  if(logado){
    cartoes.push(cartao)
    salvaCartoes()
    cartao.on("mudanca.**", render)
    preparaCartao(cartao)
    let listaImagens = Cartao.pegaImagens(cartao)
    caches.open("ceep-imagens").then(cache => {
      cache.matchAll(listaImagens).then(lista => {

        })
      })
    render()
    return true
  } else {
```

```
      alert("Você não está logado")
    }
  }
```

Voltando à aba "Application" do navegador, vamos relembrar que o cache `ceep-imagens` armazena o request (no caso, a url) e mais algumas informações, cujo valor é uma resposta. Ela guarda muito mais informações além do próprio conteúdo, que é a imagem em si. Recarregando a página com a aba "Network" aberta, tentarei mostrar as imagens. Observe que uma resposta é muito mais do que apenas seu conteúdo: há todos os cabeçalhos da resposta, e outras informações guardadas pelo Cache Storage quando solicitamos o armazenamento da imagem correspondente.

No código, portanto, a `listaImagens` não é uma lista qualquer, e sim uma lista de respostas. Aquilo a que teremos acesso a partir dela é o que chamamos de "**corpo da resposta**", o conteúdo que queremos colocar na tag `img` , na `src` . Com essa lista de respostas, para cada uma delas, preciso obter acesso ao seu código. Com isto, precisamos passá-lo para dentro da tag `src` do nosso cartão.

É preciso transformar essa `src` , que atualmente é uma url, em código da imagem. Ou seja, precisamos converter essa `src` no que chamamos de "**base64**", em corpo da resposta. Isto não é algo trivial de ser feito, como se pode ver no exemplo do meu GitHub:

Exemplo GitHub

Abro um cache através do `caches.open` , tenho uma imagem armazenada dentro dele (`"app-images"`) faço todo aquele processo com `cache.match` e tenho a resposta. Tive que ler a tag `body` e depois convertê-la para um `base64` , que é o código necessário para conversão das imagens. Recebo um `array` de bytes, então não é algo simples, não se trata de texto. Teríamos que fazer tudo isto e inserir a imagem específica dentro do cartão, ou seja, precisaríamos saber qual o cartão exato e inseri-la ali. Isto acaba dando muito trabalho, sendo que é algo que o browser deveria, e já faz por nós.

Quando coloco uma `src` igual a uma `url` , o browser já sabe o que precisa fazer: pegar uma imagem de algum lugar e disponibilizar sua visualização para o usuário. Tal `src` tenta automaticamente acessar a imagem através do servidor, e é isso que precisamos tentar evitar agora, mostrando ao navegador que quando o `src` for encontrado, ele não deve mais consultar diretamente o servidor, e sim procurar o conteúdo no Cache Storage. Caso a imagem seja encontrada ali, ela será utilizada. Fazer o navegador parar de procurar algo diretamente no servidor já foi feito antes, com o "Application Cache", o primeiro lugar a ser consultado.

Os arquivos listados ali não vêm do servidor, e sim do cache. O que estamos fazendo agora é bastante similar ao que o Application Cache fazia automaticamente. As imagens e seu conteúdo dinâmico, deveriam estar listados ali, mas é justamente isso que queremos evitar. Salvamos as imagens no Cache Storage, e agora temos que mostrar ao browser que as imagens devem ser carregadas a partir dali.

Os pedidos que o navegador fará precisam ser interceptados dos servidores, para ter acesso às imagens. O código precisa acessar a imagem solicitada pelo usuário. Por exemplo, quando a tag `img` é utilizada, ela está solicitando a respectiva url. Se tivermos acesso a esta informação, conseguimos colocar a resposta no Cache Storage. Para acessarmos o pedido do navegador, essa tecnologia que intercepta informações, o que fazemos aqui é algo muito bom, criado pela W3C: os ***Service Workers***.

Eles farão exatamente o que precisamos, que é acessar o cache a partir do qual virá a resposta. Trata-se de um código JavaScript criado em um arquivo separado.

Vamos criar um novo arquivo e chamá-lo de `service-worker.js` . Abrimos o `index.html` , tentamos carregar os `script` s. O Service Worker é um tanto diferente, por não se tratar de um JavaScript, como o jQuery ou o `Filtro.js` , por exemplo. Não é um código a ser executado quando a página é carregada, tampouco tem a ver com uma funcionalidade específica. Ele ficará

rodando junto à nossa App. Quando ela pedir algum arquivo externo, o pedido será interceptado, rodando paralelamente ao nosso site.

Iremos, portanto, registrar um Service Worker, o que quer dizer que há um `.js` registrando este arquivo `service-worker.js`. Precisarei acrescentar ao `index.html` uma linha de código responsável pelo registro desses `service-worker.js`, ou seja, por avisar o browser de que há um serviço rodando na nossa aplicação:

```
<script src="js/sw/registra.js"></script>
```

Vou à pasta "js", dentro da qual crio outra, denominada "sw", que por sua vez abriga o arquivo `registra.js`, em que salvaremos todo o código que registrará o Service Worker.

Diretorio

O primeiro item a ser registrado é o navegador:

```
navigator.serviceWorker.register('service-worker.js')
```

Com isto, o arquivo `service-worker.js` será buscado no nosso site, e registrado. Sua informação aparecerá em "Service Workers" da aba "Application". Precisamos lembrar, porém, que tudo que for alterado nos arquivos `.js` e até mesmo nos `.html` não serão atualizados no navegador de forma automática, pois estamos utilizando o Application Cache. Avisarei, então, no `offline.manifest`, que estamos usando uma nova versão ("#v3"):

v3

Recarregamos a página, esperamos que ele tenha baixado o cache, tudo certo, vou carregar mais uma vez para assegurar que as alterações foram salvas. O Service Worker foi registrado. Estamos com um erro por não termos mais uma lista de imagens (`listaImagens`), já que apenas copiei e colei o trecho necessário de outro arquivo para o `service-worker.js`.

Erro listaImagens

Ao abrirmos a parte de "Service Workers" agora, pode-se ver algumas modificações. Possuo um Service Worker dentro do arquivo em `.js` de mesmo nome, o qual está ativo e rodando perfeitamente. Se conseguirmos acertar todo o código presente em `service-worker.js`, conseguiremos interceptar todos os requests, atingindo assim nosso objetivo.

Precisamos arrumar esse código! A primeira coisa a se fazer é encontrar uma maneira de acessar a imagem solicitada pelo navegador dentro do Service Worker. Queremos executar esse código sempre que a imagem for pedida. Para isto, tentaremos acessar a url da imagem (através da função `urlImagem`). Deixaremos de trabalhar com todas as imagens dos cartões, pois cada uma delas terá uma tag `img`. Cada imagem dispara um pedido ao servidor. Então, neste novo código, tentaremos fazer com que a execução seja feita a cada pedido, para cada imagem.

No arquivo `service-worker.js`, quando estivermos com o cache de imagens aberto, não usaremos mais `matchAll`, substituo-a por `match`, uma vez que trata-se de uma imagem apenas, não de todas. A variável `urlImagem` precisa ser acessada, pois não consigo dizer qual imagem estará aqui especificamente. Esta informação precisa ser pega quando o navegador realizar o pedido.

```
let urlImagem =
caches.open("ceep-imagens").then(cache => {
  cache.match(urlImagem).then(listaResposta => {
```

```
    listaResposta
    })
  })
```

Todo este código só poderá ser executado quando o navegador fizer um pedido. Como vimos no console do navegador, o erro se deu quando o `service-worker.js` foi registrado. Todo código que fica "solto" ali é executado quando registramos este arquivo. Caso queiramos executar esse código depois, precisamos isolá-lo em uma função, a ser executada somente quando o browser estiver tentando carregar algum arquivo externo. Neste momento, a função `fetch()` é disparada, e consigo ouvir este evento. O nome "*event listener*" ou "ouvinte de eventos" é comum, sendo usado nas nossas páginas para verificação da execução de algo.

Em termos práticos, estamos falando que a função `carregaDoCacheStorage()` será executada a partir do evento `fetch()`. Para fazê-lo dentro do Service Worker, vou adicionar um *event listener* (" `self.addEventListener()` "), sendo que o nome do evento será `fetch` e o parâmetro executado, a função `carregaDoCacheStorage`.

```
self.addEventListener("fetch", carregaDoCacheStorage)

function carregaDoCacheStorage(){
  let urlImagem =
  caches.open("ceep-imagens").then(cache => {
    cache.match(urlImagem).then(listaResposta => {
      listaResposta
      })
    })
}
```

Esta função pode ser utilizada como parâmetro, ou podemos simplesmente criar a função inteira dentro dela, de forma que não precisamos nem nomeá-la. Ela passa a ser uma função anônima:

```
self.addEventListener("fetch", function (){
  let urlImagem =
  caches.open("ceep-imagens").then(cache => {
    cache.match(urlImagem).then(listaResposta => {
      listaResposta
      })
    })
  })
```

Este é o tipo de código que precisa rodar sempre que um `fetch()` for executado e um arquivo for carregado. Verificaremos se ele está sendo executado de fato através do `console.log()`:

```
self.addEventListener("fetch", function carregaDoCacheStorage(){
  console.log("Tentou carregar alguma coisa")
  cache.open("ceep-imagens").then(cache => {
    cache.match(urlImagem).then(listaResposta => {
      listaResposta
      })
    })
  })
```

Se tudo ocorrer conforme o esperado, o Service Worker estará atualizado. Vamos testar recarregando o navegador e conferindo na aba "Application":

Service Workers

Na parte "Status", ele indica que trata-se de um Service Worker número 112, dado pelo navegador. No entanto, existe outro (número 113) aguardando ativação ("waiting to activate").

O Service Worker já existia e foi atualizado, o que não acontece automaticamente. Por padrão, o que podemos fazer é: sempre que fechamos o site e o reabrimos, o Service Worker #113 é o que está ativado. E para ser ativado e atualizado, todas as outras abas precisam ser fechadas.

Agora que tudo está atualizado, conseguimos verificar seu funcionamento no console. Estamos com erros, porém, observe a mensagem recebida: "Tentou carregar alguma coisa", ou seja, ele foi executado, o problema é que ele tentou carregar todos os arquivos, incluindo todos os `.css`, `script`, cada um gerando esta mesma mensagem.

Temos um código errado ali no meio. A url da imagem não está definida, pois não temos acesso à ela ainda. Já temos arquivos que são executados conforme suas solicitações. Precisamos pegá-los do cache, o que deve ser feito através do acesso à url da imagem solicitada. Na verdade, não sei nem se é uma imagem que está sendo solicitada!

Caso tenha a url do pedido (por meio da variável `UrldoPedido`), pode-se verificar se ele se encontra no cache, acessando as informações do evento de `fetch`. Com isto, consigue-se acessar o request feito. A cada evento disparado, variam-se as informações, pois os arquivos são diferentes também. A informação é passada como parâmetro através do "Event Listener".

```
self.addEventListener("fetch", function(event){
  let urlDoPedido = event.request
  caches.open("ceep-imagens").then(cache => {
    cache.match(urlDoPedido).then(listaResposta => {
      listaResposta
      })
    })
  })
```

Em que `event.request` se refere a todas as informações do pedido, não somente a url correspondente. Estávamos pensando em acessar a informação armazenada pelo Cache Storage só pela url, que é como a aba "Application" nos apresenta a informação. Além da url, o que é armazenado é a informação do pedido (ou request). O que significa que além das informações provenientes da url, tem toda a parte dos cabeçalhos enviados, como visto anteriormente na aba "Network".

Tudo que é carregado no servidor como resposta nos traz várias informações com cabeçalhos, sendo que o `event.request` é exatamente um pedido que podemos procurar diretamente no Cache Storage. Não há problema solicitarmos um request em vez de uma url. Se o pedido foi feito e salvo antes, ele será encontrado no cache, e o que teremos como retorno é uma resposta, não mais uma lista.

```
self.addEventListener("fetch", function(event){
  let pedido = event.request
  caches.open("ceep-imagens").then(cache => {
    cache.match(pedido).then(resposta => {
      resposta
      })
    })
  })
```

O que faço, então, com a resposta em mãos? Preciso avisar ao evento de fetch que a resposta não será a mesma que for buscada no servidor. Precisamos passar a resposta para o evento, e para isto chamaremos uma função `event.respondWith(resposta)`, o qual sempre deverá ser executada antes do fim da `EventListener`.

Abrir o cache e acessar um pedido dentro dele é uma ação que demanda certo tempo, é assíncrona, não "trava o código". Se houver qualquer ação abaixo desta, não há garantia de que a `event.respondWith(resposta)` foi executada. É por esse motivo, inclusive, que utilizamos uma `Promise`. O browser não sabe que precisa esperar o Cache Storage ser aberto. É por isso que a função `event.respondWith(resposta)` não pode ser executada dentro do callback, e sim quando alcançar a função `fetch()`.

Temos uma `Promise`, e o resultado dela, no fim, é a resposta, a `event.respondWith`, que sabe que acessar a informação presente no cache é algo assíncrono, assim como acessar algo que está no servidor. Por padrão, a `event.respondWith` recebida não é a resposta, e sim uma `Promise`, que terá como resultado a resposta. Após sua nomeação, cada resposta se dá exatamente pela abertura do cache e verificação da existência de algum arquivo lá dentro.

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

  let pedido = event.request
  let promiseResposta = caches.open("ceep-imagens").then(cache => {
    cache.match(pedido).then(resposta => {
      resposta
      })
    })

    event.respondWith(promiseResposta)
  })
```

No entanto, neste instante, o resultado da `Promise` é tudo aquilo que é respondido dentro do callback, pois originalmente a `Promise` `caches.open()` tem como resultado o cache previamente aberto. Quando acrescentamos `.then()` alteramos a `Promise`, cujo resultado, agora, é aquilo que retornamos dentro dela. Quero que a resposta seja o resultado, e para tal acrescentarei a `return` ao código. Só que ela é parte da função de callback de outra `Promise`. Preciso que a `return` seja da função de fora dela, a que vem antes, a partir do request.

Para que isso ocorra, usaremos um comportamento das promises criado exatamente para este fim. Há algo assíncrono (procurar o pedido) que está dentro de outra ação assíncrona (abrir o cache). Consigo encadear esse tipo de atividade na `Promise`, dizendo "abra o cache, encontre o pedido lá dentro, e depois disto tudo, quero a resposta como resultado", ou seja, coloco o `.then()` para fora, encadeando-o.

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

  let pedido = event.request
  let promiseResposta = caches.open("ceep-imagens").then(cache => {
    return cache.match(pedido)
    }).then(resposta => {
      return resposta
      })

    event.respondWith(promiseResposta)

  })
```

Se a `Promise` for encadeada desta maneira, só fica faltando, depois de abrir o cache e recebê-lo, dizer que o que terei na próxima `Promise` é a resposta, o resultado vindo do `cache.match()`, bastando retorná-la no callback.

Ao utilizarmos o navegador, podemos ver o Service Worker funcionando, respondendo com as imagens que estiverem no Cache Storage.