第4章 Service Workerを使った表示高速化、オフライン対応について

前回、Service Workerを使ってキャッシュ登録を行いました。今回はそのキャッシュを返す処理を作成します。キャッシュを返すことによって二つのメリットが生まれます。

- サーバにアクセスしないので表示が高速
- サーバにアクセスしないのでオフラインでも表示できる

CACHE APIはローカルプロキシと比喩されることがありますが、リモートサーバにアクセスする前にCACHE APIを呼び出すので、まさにプロキシとして利用できます。

キャッシュ処理を確認する

二回目以降のWebブラウザでのアクセスはすべてService Workerの `fetch` イベントを通過します。`event.respondWith` はHTTPレスポンスを受け取ってWebブラウザに返します。`event.request` の中にリクエスト情報が入っています。sw.jsの中のfetchイベントを探して、下記のように変更してください。

元：

```
// ネットワークアクセス時に使われるfetchイベント
self.addEventListener('fetch', async (event) => {
});
```

修正後：

```
// ネットワークアクセス時に使われるfetchイベント
self.addEventListener('fetch', async (event) => {
  event.respondWith(
    fetch(event.request)
      .then(response => {
        return response;
      },
      error => {
        return caches.match(event.request)
      });
    )
});
```

ここで注意して欲しいのは fetch イベントはキャッシュに登録したURL以外の場合も呼ばれるということです。そこでキャッシュの中から該当リクエストがあるかどうかを確認します。そのレスポンスがあればそのまま返し、なければFetch APIを使ってHTTPリクエストを行います。

オフラインの場合どうなるか？

オフラインの場合、挙動はどうなるでしょうか。そのURLをキャッシュしている場合、レスポンスがあるのでキャッシュを返します。つまりオフライン対応できます。対してレスポンスがない（キャッシュしていない）場合は Fetch APIを使ってHTTPリクエストを行いますが、オフラインなので当然失敗します。そのためオフラインの場合にはキャッシュがあれば表示され、なければ表示できないという状態になります。当たり前ですが、オフライン時にどういう表示にするかは開発者に委ねられていることになります。

ここまででTodoアプリのオフライン化、表示高速化が実現しました。次にTodoの表示処理をオフライン対応させてみましょう。