Le cache HTTP

Date : 13 avril 2023
Speakers : Hubert Sablonnière (Clever Cloud)
Format : Conférence (45mn)
GitHub : https://github.com/hsablonniere/talk-back-to-basics-cache/tree/version-2023

Introduction

Activité : quoi regarder sur Netflix ?
Dans les années 90, on allait au vidéo club. On regardait souvent le même film et on devait faire l'aller-retour vers le vidéo-club.

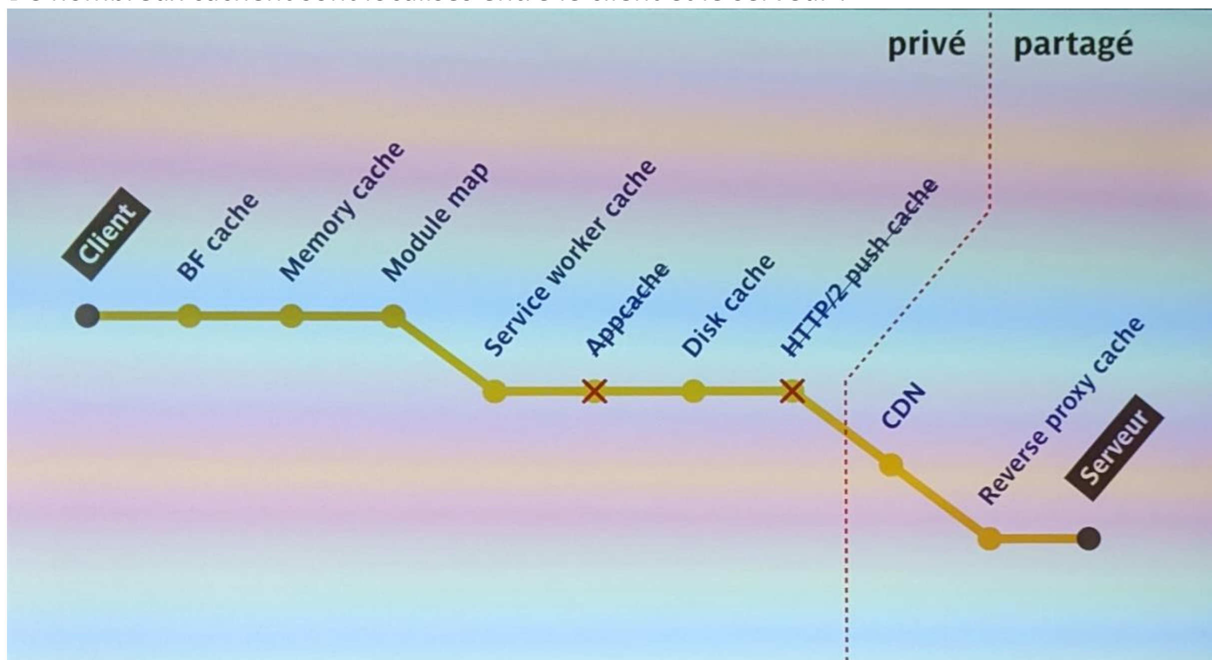
Mai 1996 : T. Berners-Lee, R. Fielding, UC Irvie et H. Frystyk publient la RFC 1945 sur HTTP/1.0 avec la première notion de cache HTTP. Le principe est simple : entre le client et le serveur, le cache rapproche la source de la réponse. Le cache présente 3 avantages :

1. Permet de réduire le chargement côté client
2. Réduit la charge côté serveur
3. Meilleures perfs = meilleur business

Le cache c'est la vie mais c'est très compliqué. Qui n'a jamais dit ou entendu dire : « Le bug est corrigé, mais il faut que tu vides ton cache … » ?

Derrière le cache HTTP se cache une histoire d'en-têtes et de nommage des sources entre le frontend et le backend.

De nombreux cachent sont localisés entre le client et le serveur :



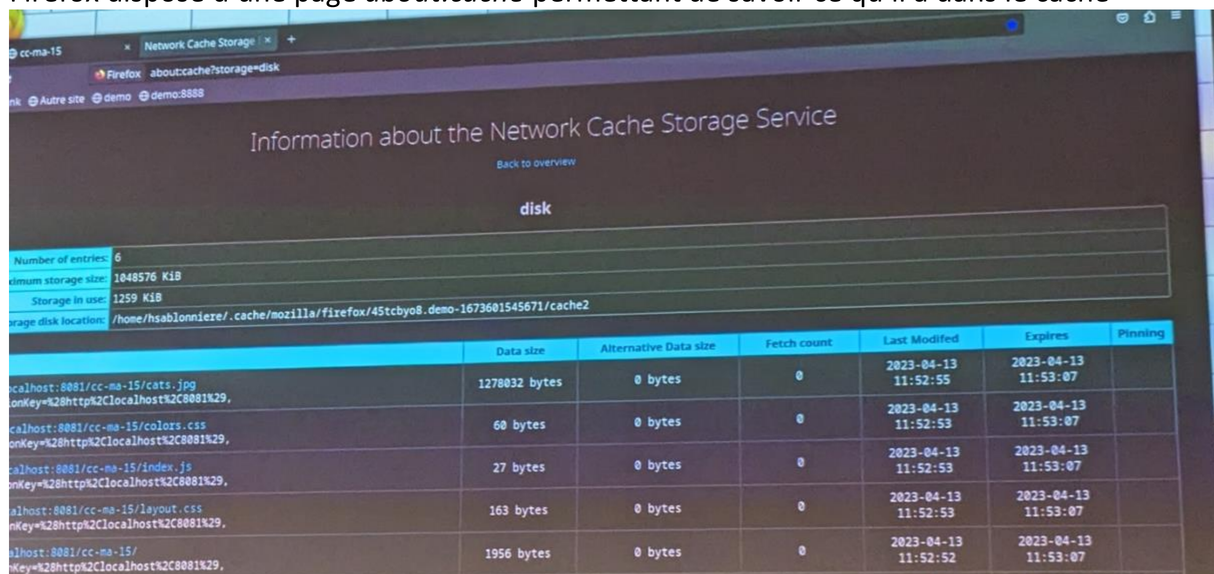
Cache navigateur

<u>Requête</u> :
GET /index.html http/1.1

<u>Réponse</u> :
HTTP/1.1 200 OK
cache-control: max-age=15

Le cache n'est pas impératif, mais déclaratif. On a le droit d'utiliser la donnée pendant X secondes. Après, la donnée est périmée.

Démo avec un serveur Node JS et un navigateur Firefox.
cache-control: max-age=15
Le navigateur n'envoie des requêtes au serveur qu'après 15 secondes.
Firefox dispose d'une page *about:cache* permettant de savoir ce qu'il a dans le cache



Le navigateur peut faire une demande de revalidation avec le serveur.

2 mécanismes sont proposés aux développeurs pour effectuer des requêtes conditionnelles :

1. **Etag** : dans la réponse HTTP on a une en-tête *etag* représentant un hash ou une version. Lors d'une requête on peut utiliser le *if-none-match : « etag ».* Le serveur renvoie uniquement une réponse 304 sans toute la donnée
2. **Last-modified** : stratégie possible lorsqu'on connait la date de mise à jour d'une page comme dans un CMS. En-têtes *last-modifier* et *if-modified-since*. Démo avec date de modification d'il y'a 10 heures. La date de péremption est calculée par Firefox à l'aide d'une règle de 10%. Bonne pratique : forcer la validation avec *no-cache*

Attention, la directive *cache-control : no-cache* ne veut pas dire pas de cache. Droit de mettre en cache, mais force la revalidation. On retrouve le comportement similaire au etag.

Cache-control : no-store => force le navigteur à ne pas utiliser le cache

Cache-control : max-age=60, must-revalidate => s'utilise avec *max-age*, revalide si la ressource est périmée lorsque le serveur envoie des données périmées. Pas de démo.

Réponse :
Cache-control : max-age=31588888, immutable

Pas de revalidation en cas de rechargement
Démo depuis Safari qui supporte le *immutable*. Le navigateur ne recharge pas les sous-ressources avec F5. Le *immutable* ne fonctionne pas dans Chrome, mais ce dernier l'implémente par défaut. Firefox adopte le comportement de Chrome.

Pour être certain que la ressource ne soit pas récupérée d'un cache, il est possible de mettre une empreinte dans le nom du fichier (via des outils comme webpack)
GET /index.sdALn45Sds6ks29d.html

Caches partagés : cache des reverses proxy et CDN

Grâce aux caches partagés, un second utilisateur profite du cache partagé remplit par le premier utilisateur.

Réponse :
age : 120 => ajouté par le reverse proxy et le CDN pour indiquer au navigateur depuis combien de temps la ressource est dans son cache

Caches privés uniquement
Cache-control : …., **private**

La RFC dit que les requêtes utilisant l'en-tête *authorization* ne doit être stocké que dans des caches privés.

Spécificités des caches partagés
Nouvelles spécifications en 2022 pour cibler des directives cache-control
Exemples :
cdn-cache-control:
akamai-cache-control:

L'en-tête **vary** est dangereuse. Démo avec 3 navigateurs : Firefox, Chromium et Safari.
Reverse proxy Caddy
Hugo bascule sur le port du reverse proxy
Problème avec la négociation de contenu et la langue indiquée dans l'en-tête A*ccept-language*
vary permet de parer à ce problème mais dans le cache on a autant de valeur qu'il y'a de langues
Bon cas d'usage de vary : pour faire de la compression de données

Jusqu'ici on parlait du disk cache du navigateur. Il y'a également le memory cache qui rentre en jeu. Lorsqu'Hugo reste sur le même site, Firefox et Chrome utilisent le cache mémoire.

Module map

Les modules ECMAScript de 2 scripts JS viennent du même module map => chargé une seule fois.

http/2 push cache

Un peu has been
Personne n'utilise ce cache et va être remplacer par les Early Hints

App cache

Déprécié

Service Worker cache

Proxy programmé en JS.

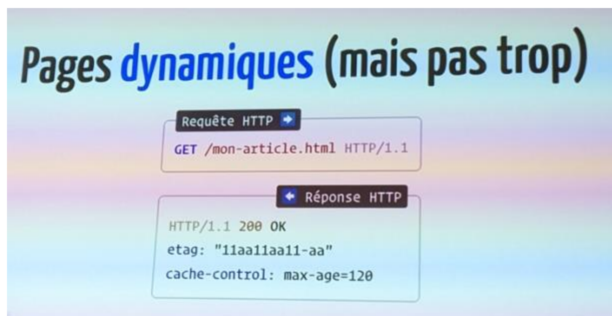
Partionnement de cache

Firefox stocke 2 fois la même image qui est chargée depuis 2 sites différents

Backward/Forward cache

Utilisé lors des navigations précédents / suivants
Firefox : lors d'un précédent, la page est congelée ; les données saisies, position du scroll, curseur, … sont restorées

Quelques recettes pour conclure



Pas de cache

Requête HTTP

```
GET /index.html HTTP/1.1
```

Réponse HTTP

```
HTTP/1.1 200 OK
cache-control: no-store
```

Si ça se compresse

Requête HTTP

```
GET /index.html HTTP/1.1
accept-encoding: gzip, deflate, br
```

Réponse HTTP

```
HTTP/1.1 200 OK
content-encoding: br
vary: accept-encoding
```

Si c'est spécifique au profil connecté

Requête HTTP

```
GET /profile.html HTTP/1.1
cookie: session-id=42
```

Réponse HTTP

```
HTTP/1.1 200 OK
cache-control: [...], private
```