RFC 7234 : Hypertext Transfer Protocol (HTTP/1.1): Caching

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 15 juin 2014

Date de publication du RFC : Juin 2014

https://www.bortzmeyer.org/7234.html

Le protocole HTTP 1.1, désormais décrit dans une série de RFC <https://www.bortzmeyer.org/http-11-reecrit.html>, transporte énormément de données tous les jours et consomme donc à lui seul une bonne partie des ressources de l'Internet. D'où l'importance de l'optimiser. Une des méthodes les plus efficaces pour cela est le **cache** (terme anglais qui fait très bizarre en français : pour mieux accéder à une ressource, on la cache...). Ce RFC spécifie le modèle de cachage de HTTP et comment les clients et les serveurs peuvent l'utiliser. (Il a depuis été remplacé par le RFC 9111 [1].)

Un cache Web est un espace de stockage local où on peut conserver la représentation d'une ressource qu'on a récupérée. Si la même ressource est à nouveau désirée, on pourra la récupérer depuis le cache, plus proche et donc plus rapide que le serveur d'origine. Outre le temps d'accès, le cachage a l'avantage de diminuer la consommation de capacité réseau. Un cache peut être partagé entre plusieurs utilisateurs, augmentant ainsi les chances qu'une ressource désirée soit présente, ce qui améliore l'efficacité. Comme tous les caches, les caches Web doivent gérer le stockage, l'accès et la place disponible, avec un mécanisme pour gérer le cas du cache plein. Comme tous les caches, les caches Web doivent aussi veiller à ne servir que de l'information fraîche. Cette fraîcheur peut être vérifiée de différentes façons, y compris par la validation (vérification auprès du serveur d'origine). Donc, même si l'information stockée dans le cache n'est pas garantie fraîche, on pourra quand même l'utiliser, si le serveur d'origine confirme qu'elle est toujours utilisable (dans ce cas, on aura quand même un accès réseau distant à faire, mais on évitera de transférer une ressource qui peut être de grande taille).

Le cache est optionnel pour HTTP, mais recommandé, et utiliser un cache devrait être le comportement par défaut, afin d'épargner le réseau, pour lequel HTTP représente une bonne part du trafic.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc9111.txt

On peut garder en cache plusieurs sortes de réponses HTTP. Bien sûr, le résultat d'une récupération après un `GET` (code 200, cf. RFC 7231) est cachable et représente l'utilisation la plus courante. Mais on peut aussi conserver dans le cache le résultat de certaines redirections, ou bien des résultats négatifs (un code 410, indiquant que la ressource est définitivement partie), ou même le résultat de méthodes autres que `GET` (bien que cela soit plus rare en pratique).

Ici, un exemple où une page a été stockée par un cache Squid, et récupérée ensuite. L'argument de `GET` est l'URI complet, pas juste le chemin :

```
% curl -v http://www.w3.org/WAI/
...
> GET http://www.w3.org/WAI/ HTTP/1.1
> User-Agent: curl/7.26.0
> Host: www.w3.org
...
< HTTP/1.0 200 OK
< Last-Modified: Thu, 12 Jun 2014 16:39:11 GMT
< ETag: "496a-4fba6335209c0"
< Cache-Control: max-age=21600
< Expires: Sun, 15 Jun 2014 15:39:30 GMT
< Content-Type: text/html; charset=utf-8
< Age: 118
< X-Cache: HIT from cache.example.org
< X-Cache-Lookup: HIT from cache.example.org:3128
< Via: 1.1 cache.example.org:3128 (squid/2.7.STABLE9)
...
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Le client HTTP curl suit la variable d'environnement `http_proxy` et contacte donc le relais/cache Squid en `cache.example.org` en HTTP. À son tour, celui-ci se connectera au serveur d'origine si nécessaire (ce ne l'était pas ici, l'information a été trouvée dans le cache, comme l'indique la mention *"HIT"*.)

Les données stockées dans le cache sont identifiées par une **clé** (section 2 de notre RFC). Pour un cache simple, qui ne gère que `GET`, la clé principale est l'URI de la ressource convoitée. On verra plus loin que la clé est en fait plus complexe que cela, en raison de certaines fonctions du protocole HTTP, comme la négociation de contenu `<https://www.bortzmeyer.org/negociation-contenu-http.html>`, qui impose d'utiliser comme clé certains en-têtes de la requête.

La section 3 du RFC normalise les cas où le cache a le droit de stocker une réponse, pour réutilisation ultérieure. Le RFC définit ces cas négativement : le cache ne doit **pas** stocker une réponse **sauf si** toutes ces conditions sont vraies :

- La méthode est cachable (c'est notamment le cas de `GET`),
- Le code de retour est compris du cache (200 est le cas évident),
- Il n'y a pas de directive dans la réponse qui interdise le cachage (en-tête `Cache-Control:`, voir plus loin),
- L'accès à la ressource n'était pas soumis à autorisation (cf. RFC 7235), dans le cas d'un cache partagé entre plusieurs utilisateurs,
- La réponse contient des indications permettant de calculer la durée de vie pendant laquelle elle restera fraîche (comme l'en-tête `Expires:`).

Un cache peut stocker des réponses partielles, résultat de requêtes avec intervalles (cf. RFC 7233), si lui-même comprend ces requêtes. Il peut concaténer des réponses partielles pour ensuite envoyer une ressource complète.

Une fois la ressource stockée, le cache ne doit **pas** la renvoyer **sauf si** (là encore, la norme est formulée de manière négative, ce qui est déroutant) toutes ces conditions sont vraies :

— Les URI correspondent,
— Les en-têtes désignés par l'en-tête `Vary:` correspondent (cela concerne surtout le cas où il y a négociation du contenu),
— La requête ne contient pas de directive interdisant de la servir avec des données stockées dans le cache,
— La ressource stockée est encore fraîche, ou bien a été re-validée avec succès.

L'exigence sur la clé secondaire (les en-têtes sur lesquels se fait la négociation de contenu) est là pour s'assurer qu'on ne donnera pas à un client une ressource variable et correspondant aux goûts d'un autre client. Si le client dont la requête a déclenché la mise en cache avait utilisé l'en-tête `Accept-Language: fr` indiquant qu'il voulait du français, et que le second client du cache demande la même ressource, mais avec `Accept-Language: en`, il ne faut évidemment pas donner la copie du premier client au second. Si la réponse avait l'en-tête `Vary: accept-language` indiquant qu'elle dépend effectivement de la langue, le cache ne doit la donner qu'aux clients ayant le même `Accept-Language:`.

Et la fraîcheur, elle se définit comment (section 4.2, une des plus importantes du RFC) ? Le cas le plus simple est celui où le serveur d'origine envoie un en-tête `Expires` (ou une directive `max-age`), par exemple `Expires: Mon, 15 Jun 2015 09:33:06 GMT` (un an dans le futur). Dans ce cas, la ressource gardée en cache est fraîche jusqu'à la date indiquée. Attention : les formats de date de HTTP sont compliqués et il faut être prudent en les analysant. Si le `Expires:` indique une date syntaxiquement incorrecte, le cache doit supposer le pire et considérer que la ressource a déjà expiré. En pratique, bien des serveurs HTTP ne fournissent pas cet en-tête `Expires:` et le cache doit donc compter sur des heuristiques. La plus courante est d'utiliser le champ `Last-Modified:` et de considérer que, plus le document est ancien, plus il restera frais longtemps (section 4.2.2). (La FAQ `<http://wiki.squid-cache.org/SquidFaq/InnerWorkings#How_does_Squid_decide_when_to_refresh_a_cached_object.3F>` de Squid explique bien l'heuristique de ce logiciel de cache.) Le RFC ne normalise pas une heuristique particulière mais met des bornes à l'imagination des programmeurs : ces heuristiques ne doivent être employées que s'il n'y a pas de date d'expiration explicite, et la durée de fraîcheur doit être inférieure à l'âge du document (et le RFC suggère qu'elle ne soit que 10 % de cet âge). Par contre, l'ancienne restriction du RFC 2616 contre le fait de garder en cache des documents dont l'URL comprenait des paramètres (après le point d'interrogation) a été supprimée. Un serveur ne doit donc pas compter que ces documents seront forcément re-demandés. Il doit être explicite quant à la durée de vie de ces documents.

Dans sa réponse, le cache inclut un en-tête `Age:`, qui peut donner au client une idée de la durée depuis la dernière validation (auprès du serveur d'origine). Par exemple, `Age: 118`, dans le premier exemple, indiquait que la page était dans le cache depuis presque deux minutes.

Une réponse qui n'est pas fraîche peut quand même être renvoyée au client dans certains cas, notamment lorsqu'il est déconnecté du réseau et ne peut pas donc valider que sa copie est toujours bonne. Le client peut empêcher cela avec `Cache-Control: must-revalidate`.

Comment se fait cette validation dont on a déjà parlé plusieurs fois ? Lorsque le serveur a une copie d'une ressource, mais que sa date maximum de fraîcheur est dépassée, il peut demander au serveur d'origine. Cela se fait typiquement par une requête conditionnelle (cf. RFC 7232) : si le serveur a une copie plus récente, il l'enverra, autrement, il répondra par un 304, indiquant que la copie du cache est bonne. La requête conditionnelle peut se faire avec un `If-Modified-Since:` en utilisant comme date celle qui avait été donnée dans le `Last-Modified:`. Ou bien elle peut se faire avec l'"*entity tag*" et un `If-None-Match:` :

```
% telnet cache 3128
...
GET http://www.w3.org/WAI/ HTTP/1.1
Host: www.w3.org
If-None-Match: "496a-4fba6335209c0"

HTTP/1.0 304 Not Modified
Date: Sun, 15 Jun 2014 09:39:30 GMT
Content-Type: text/html; charset=utf-8
Expires: Sun, 15 Jun 2014 15:39:30 GMT
Last-Modified: Thu, 12 Jun 2014 16:39:11 GMT
ETag: "496a-4fba6335209c0"
Age: 418
X-Cache: HIT from cache.example.org
X-Cache-Lookup: HIT from cache.example.org:3128
Via: 1.0 cache.example.org:3128 (squid/2.7.STABLE9)
Connection: close
```

Le cache peut aussi utiliser la méthode `HEAD` pour tester sa copie locale auprès du serveur d'origine, par exemple pour invalider la copie locale, sans pour autant transférer la ressource.

La section 5 liste tous les en-têtes des requêtes et des réponses qui sont utilisés pour le bon fonctionnement des caches, comme `Age:`, `Expires:`, etc. Ils sont enregistrés à l'IANA, dans le registre des en-têtes <https://www.iana.org/assignments/message-headers/message-header-index.html>. Notons parmi eux le peu connu (et peu mis en œuvre, au point que ce RFC supprime plusieurs possibilités avancées de cet en-tête) `Warning:` qui permet de transférer des informations utiles au cache mais qui ne tiennent pas dans les autres en-têtes. Par exemple, un serveur peut indiquer qu'il ne sera pas joignable (pour revalidation), avec le code d'avertissement 112 (ces codes sont dans un registre spécial <https://www.iana.org/assignments/http-warn-codes>):

```
HTTP/1.1 200 OK
Date: Sat, 25 Aug 2012 23:34:45 GMT
Warning: 112 - "network down" "Sat, 25 Aug 2012 23:34:45 GMT"
```

Parmi ces en-têtes, `Cache-Control:` (autrefois `Pragma:`) permet de spécifier des directives concernant le cache. Un client d'un cache peut spécifier l'âge maximum qu'il est prêt à accepter (directive `max-age`), une fraîcheur minimum (directive `min-fresh`), que la ressource ne doit pas être stockée dans le cache (directive `no-store`, qui est là pour des raisons de vie privée mais, bien sûr, est loin de suffire pour une véritable confidentialité), ou bien qu'elle peut être stockée mais ne doit pas être servie à un client sans revalidation (directive `no-cache`), etc. Les directives possibles sont stockées dans un registre IANA <https://www.iana.org/assignments/http-cache-directives>.

L'en-tête `Cache-Control:` peut aussi être utilisé dans des réponses. Un serveur peut lui aussi indiquer `no-cache`, typiquement parce que ce qu'il envoie change fréquemment et doit donc être revalidé, `private` s'il veut insister sur le fait que la réponse n'était destinée qu'à un seul utilisateur et ne doit donc pas être transmise à d'autres, etc.

À noter qu'un cache HTTP n'est pas forcément un serveur spécialisé. Tous les navigateurs Web ont des fonctions d'historique (comme le bouton "*Back*"). Est-ce que celles-ci nécessitent des précautions analogues à celles des caches, pour éviter que le navigateur ne serve des données dépassées ? Pas forcément, dit le RFC, qui autorise un navigateur à afficher une page peut-être plus à jour lorsqu'on utilise le retour en arrière dans l'historique.

La section 8 détaille les problèmes de sécurité qui peuvent affecter les caches. Un cache, par exemple, peut permettre d'accéder à une information qui n'est plus présente dans le serveur d'origine, et donc de rendre plus difficile la suppression d'une ressource. Un cache doit donc être géré en pensant à ces risques. Plus grave, l'empoisonnement de cache : si un malveillant parvient à stocker une fausse représentation d'une ressource dans un cache (avec une longue durée de fraîcheur), tous les utilisateurs du cache recevront cette information au lieu de la bonne. Un cache peut avoir des conséquences pour la vie privée : en demandant une ressource à un cache partagé, un utilisateur peut savoir, à partir du temps de chargement et d'autres informations envoyées par le cache, si un autre utilisateur avait déjà consulté cette page.

L'annexe A liste les différences depuis le texte précédent, celui du RFC 2616, section 13. Elles sont nombreuses (le texte a été réécrit) mais portent surtout sur des détails, par exemple des précisions sur les conditions exactes dans lesquelles on peut garder une information dans le cache. Depuis, un autre changement a eu lieu avec le remplacement de notre RFC par le RFC 9111.

Notez que le RFC 8246 a ajouté une extension à `Cache-Control:`, pour indiquer l'immuabilité d'une ressource.