RFC 7232 : Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 14 juin 2014

Date de publication du RFC : Juin 2014

https://www.bortzmeyer.org/7232.html

Dans la longue série des RFC <https://www.bortzmeyer.org/http-11-reecrit.html> sur le protocole HTTP 1.1, ce document relativement court se consacre aux requêtes HTTP conditionnelles : quand un client dit à un serveur « envoie-moi le contenu de cette ressource mais seulement si telle ou telle condition est vraie ». La principale utilité de ces requêtes conditionnelles est d'économiser la capacité réseau si le client a déjà une copie du contenu. Une autre utilité est de permettre la mise à jour d'un document si et seulement si il n'a pas été modifié depuis que le client a commencé à travailler dessus, évitant ainsi le problème de la « mise à jour perdue ». Il est maintenant dépassé par le RFC 9110[1].

Pour indiquer ces conditions, le client ajoute un ou plusieurs en-têtes (RFC 7231) à sa requête. Quels sont les éléments dont dispose le client pour indiquer les conditions ? On les nomme les validateurs. Deux validateurs sont très répandus, la date de dernière modification d'une ressource et l'étiquette de ressource ("*entity tag*" ou "*ETag*"). D'autres peuvent être définis, par exemple par WebDAV (RFC 4918). Un validateur est fort ou faible. Un validateur fort est un validateur dont la valeur change à chaque changement, même trivial, du contenu d'une ressource (les étiquettes sont dans ce cas, comme le sont les identifiants de "*commits*" d'un VCS comme git ou Subversion). Avec du contenu Web statique, et s'il n'y a pas de négociation du contenu, une façon simple de générer un validateur fort est de condenser le contenu de la ressource, et d'utiliser le condensat comme étiquette. Les validateurs forts sont idéaux (ils identifient précisement un état d'une ressource) mais souvent difficiles, voire impossibles, à générer. Au contraire, les validateurs faibles sont moins précis (plusieurs états d'une ressource peuvent correspondre à la même valeur du validateur) mais simples à générer. Par exemple, une date de dernière modification est un validateur faible : si sa résolution est d'une seconde, et que deux changements de la ressource sont faits dans la même seconde, le validateur ne changera pas. HTTP permet les deux, chacun ayant

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc9110.txt

son utilité. Mais le développeur qui se sert des validateurs pour son application a fortement intérêt à connaître la différence ! Le RFC donne l'exemple de rapports météorologiques : pour éviter qu'un rapport reste dans un cache Web (RFC 7234) alors qu'une version plus récente existe sur le serveur d'origine, le gérant de cette base de rapports doit veiller à ce que les validateurs changent quand le contenu change. D'une manière générale, les validateurs faibles sont acceptables quand on veut juste optimiser les coûts (cas des caches) mais lorsqu'on veut modifier un contenu et éviter l'effet « perte de mise à jour », il faut utiliser un validateur fort.

Comment un serveur HTTP communique-t-il un validateur au client ? Via les en-têtes de la réponse (section 2.2 du RFC). Il y a deux en-têtes possibles :

— `Last-Modified:` qui indique la date de dernière modification de la ressource,
— `ETag:` qui indique une étiquette identifiant une version spécifique de la ressource.

Le RFC prescrit à tout serveur HTTP d'envoyer systématiquement un `Last-Modified:` dès lors qu'il peut déterminer la date de modification (si la ressource est assemblée à partir de plusieurs éléments, c'est le plus récent qui compte). Cela permettra d'utiliser plus souvent les caches et donc de diminuer la consommation de capacité réseau. Un exemple de cet en-tête (la dernière ligne) :

```
% curl -v http://verite/
> GET / HTTP/1.1
...
< HTTP/1.1 200 OK
< Server: nginx/1.6.0
< Date: Thu, 29 May 2014 19:45:17 GMT
< Content-Type: text/html
< Content-Length: 612
< Last-Modified: Thu, 24 Apr 2014 14:14:52 GMT
```

Dans la grande majorité des cas, le `Last-Modified:` est un validateur faible. (Ne serait-ce que parce que la ressource peut changer deux fois en une seconde, ce qui ne changera pas le `Last-Modified:`.)

Le `ETag:`, lui, est un identificateur opaque. Il est en général plus difficile à générer pour le serveur mais bien plus fiable (il évite notamment le problème des « deux changements dans la même seconde »). En général, c'est un validateur fort. Un exemple :

```
% curl -v http://verite/toto.txt
...
> GET /toto.txt HTTP/1.1
...
< HTTP/1.1 200 OK
< Server: nginx/1.6.0
...
< ETag: "53878f56-5"
```

Comme c'est un identificateur opaque, seul le serveur sait comment il a été généré. Premier critère avant de choisir une méthode de génération : garantir que deux versions différentes d'une ressource auront deux "*ETag*" différents. Deux méthodes courantes pour le générer sont l'utilisation d'un numéro de révision interne (par exemple si les ressources sont stockées dans un VCS) ou bien l'utilisation d'une fonction de condensation. Là encore, le RFC demande que le serveur envoie une étiquette si possible, sauf s'il y a de bonnes raisons, par exemple de performance. Donc, un serveur qui suit le RFC de près enverra les deux validateurs, la date et l'étiquette, en réponse à chaque requête.

Comment le client utilise-t-il ce validateur (section 3) ? Il dispose d'un choix d'en-têtes à mettre dans la requête permettant de dire au serveur « agis sur la ressource si et seulement si telle condition portant sur les validateurs est vraie ». Par exemple, l'en-tête `If-Match:` indique une condition portant sur l'étiquette :

```
If-Match: "4149d-88-4b1795d0af140"
```

La condition ci-dessus, dans une requête `GET`, `POST` ou autre, signifie au serveur de n'agir que si la ressource correspond à l'étiquette indiquée. Pour une méthode modifiant l'état de la ressource (comme `POST` ou `PUT`), cela permet d'éviter le syndrome de la mise à jour perdue. Voici un exemple, n'utilisant **pas** les requêtes conditionnelles :

— Le client récupère avec `GET` une ressource,
— Il la modifie localement,
— Pendant ce temps, la ressource a été modifiée sur le serveur, peut-être par un autre client,
— Le client veut enregistrer sa version et fait un `PUT`. Les modifications de l'étape précédente sont donc perdues.

Avec les requêtes conditionnelles, on aurait eu :

— Le client récupère avec `GET` une ressource, et obtient un validateur fort, l'étiquette `e68bd4cc10e12c79ff830b0ec8`
— Il la modifie localement,
— Pendant ce temps, la ressource a été modifiée sur le serveur, peut-être par un autre client,
— Le client veut enregistrer sa version et fait un `PUT` en ajoutant un `If-Match: "e68bd4cc10e12c79ff830b0ec82`
Le serveur calcule l'étiquette (ici, c'est un condensat MD5), voit qu'elle ne correspond pas, et refuse le `PUT` avec un code 412. Les modifications de l'étape précédente ne sont pas perdues.

`If-Match:` est surtout utile dans les opérations modifiant la ressource comme `PUT`. Son opposé `If-None-Match:` sert plutôt pour les `GET` lorsqu'un cache dit à un serveur « envoie-moi une copie si elle est différente de celle que j'ai déjà ». Notez que `If-None-Match:` peut prendre comme valeur une liste de "*ETags*".

Il y a aussi des pré-conditions qui portent sur la date de modification et non plus sur l'étiquette. Ce sont `If-Modified-Since:` et `If-Unmodified-Since:`. Si on envoie :

```
If-Modified-Since: Mon, 26 May 2014 19:43:31 GMT
```

dans une requête `GET`, on ne recevra le contenu de la ressource que s'il est plus récent que le 26 mai 2014. Autrement, on aura un 304, indiquant que le contenu n'a pas été changé depuis. C'est ainsi qu'un cache peut s'assurer que la copie qu'il détient est toujours valable, sans pour autant consommer de la capacité réseau. C'est également utile pour un "*crawler*", par exemple celui d'un moteur de recherche qui ne veut pas récupérer et indexer un contenu qu'il connait déjà. Le cache qui a reçu un `Last-Modified:` au dernier `GET` conserve la valeur de ce `Last-Modified:` et la renvoie dans un `If-Modified-Since:` la fois suivante. curl a des options pour cela. Ici, un script de téléchargement qui trouve dans un fichier la date de dernière modification, et ne télécharge que si le fichier est plus récent :

```
ltr_date=`head -n 1 ${LTR_LOCAL} | cut -d" " -f2`
# Allow time to elapse. The date of the file at IANA is often the day after
# the date written in the LTR. Heuristically, we add one day and a few hours.
current_date=`date +"%Y%m%d %H:%M:%S" --date="${ltr_date} +1 day +4 hour"`
...
curl --time-cond "${current_date}" ...
```

Ces en-têtes sont enregistrés à l'IANA <`https://www.iana.org/assignments/message-headers/message-header-index.html`>. Leur usage est facultatif pour le serveur HTTP et, par exemple, nginx ignore ces en-têtes, limite signalée depuis pas mal de temps <`http://trac.nginx.org/nginx/ticket/242`>.

La section 4 liste les deux codes de retour HTTP en rapport avec ces requêtes conditionnelles. 304 indique que le contenu n'a pas été modifié depuis la date donnée et le serveur redirige (d'où le 3xx) le client vers sa copie locale pré-existante. 412 indique qu'une pré-condition nécessaire n'est pas vraie. C'est en général le résultat d'une requête avec `If-Match:` lorsque l'étiquette ne correspond plus au contenu de la ressource. Ces deux codes sont dans le registre IANA <https://www.iana.org/assignments/http-status-codes>.

Les sections 5 et 6 précisent l'ordre d'évaluation des pré-conditions, entre elles, et par rapport aux autres critères de recherche. Notamment, les pré-conditions ne sont pas utilisées si la ressource n'existe pas ou si son accès est interdit. Si on a un 404 (ressource non trouvée) sans pré-conditions, on aura le même 404 avec pré-conditions, l'existence de la ressource est testée avant les pré-conditions.

Et les pré-conditions entre elles, puisqu'on peut en avoir plusieurs dans une requête ? Le serveur doit évaluer dans cet ordre (en supposant à chaque fois que l'en-tête en question soit présent ; sinon, on saute au test suivant) :

— D'abord, `If-Match:`, car il faut avant tout éviter la mise à jour perdue,
— Ensuite `If-Unmodified-Since:` (qui passe après car les dates sont moins fiables que les étiquettes),
— Ensuite `If-None-Match:` (il sert à la validation des caches, ce qui est moins crucial que d'empêcher la mise à jour perdue),
— Et enfin `If-Modified-Since:`.

Et, pour finir, quelques considérations de sécurité, en section 8. D'abord, les validateurs ne sont pas des mécanismes de contrôle d'intégrité, comme peut l'être une signature numérique (le serveur peut mentir, ou un tiers situé sur le trajet a pu modifier les en-têtes en vol). Ensuite, les étiquettes, les "*ETag*" peuvent poser des problèmes de protection de la vie privée : un serveur méchant peut générer une étiquette unique pour un client donné ce qui, lorsque le client reviendra et enverra des `If-Match:` ou `If-None-Match:` permettra de le reconnaître. Une sorte de "*cookie*" caché, donc. Un navigateur Web doit donc les traiter comme tel (oublier les étiquettes lorsque l'utilisateur vide le stock des "*cookies*", par exemple).

L'annexe A dresse une liste des différences par rapport aux sections correspondantes du RFC 2616. Rien de fondamental mais la définition de la force et de la faiblesse des validateurs a été étendue et précisée, et une définition de l'ordre d'évaluation des pré-conditions a été ajoutée.