RFC 5789 : PATCH Method for HTTP

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 19 mars 2010

Date de publication du RFC : Mars 2010

https://www.bortzmeyer.org/5789.html

Le célébrissime protocole HTTP (décrit dans le RFC 7230 [1] et suivants) prévoit plusieurs **méthodes** pour interagir avec une **ressource**, identifiée par un URI. Dans le cas le plus connu, la ressource est une page en HTML et la méthode utilisée est `GET`, pour récupérer ladite page. Mais HTTP dispose d'autres méthodes, comme `PUT` ou `DELETE`. Ce RFC 5789 ajoute à la liste une méthode de modification partielle d'une ressource, `PATCH`.

Si `GET` sert à récupérer une ressource (par exemple un fichier), `DELETE` à détruire une ressource, et `PUT` à installer une nouvelle version toute neuve, `PATCH` permettra de modifier une ressource sans envoyer l'intégralité de la nouvelle ressource. L'intérêt principal est de transmettre des mises à jour de petite taille, au lieu de renvoyer une (peut-être très grosse) ressource complète. Pour l'instant, je ne connais pas de mise en œuvre disponible officiellement (voir à la fin de cet article pour quelques idées).

Il faut noter tout de suite que le **format** de la modification envoyée n'est pas spécifié : plusieurs formats seront possible, le client HTTP devra indiquer le format utilisé et espérer que le serveur l'accepte.

Pourquoi ne pas utiliser les méthodes existantes au lieu d'inventer `PATCH` ? La section 1 répond à cette question. `PUT` remplace complètement la ressource (et utiliser `PUT` pour envoyer une modification partielle, même si le serveur final comprend l'opération, sémerait la confusion chez les relais) et `POST` est trop mal spécifié (il sert un peu de méthode fourre-tout).

La nouvelle méthode est décrite en détail en section 2. Le client envoie le "*patch*" et indique l'URI de la ressource auquel il s'applique. Le "*patch*" est une série d'instructions indiquant comment modifier l'ancienne ressource pour atteindre l'objectif, la nouvelle ressource. Il existe plusieurs langages pour de telles instructions comme les "*patches*" XML du RFC 5261, comme le langage de "*patch*" traditionnel des

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc7230.txt`

programmeurs (type non officiel `text/x-diff`), etc. Au fur et à mesure de leur définition officielle, ces types seront enregistrés dans le registre <https://www.iana.org/assignments/media-types/application/>.

Attention, comme rappelé à plusieurs reprises dans le RFC, `PATCH` n'est pas idempotent (pas plus que `POST`, voir le RFC 7231, section 9.1.2). La question a été vigoureusement discutée à l'IETF lors de l'élaboration de ce RFC et la conclusion générale était qu'assurer une telle sémantique serait très coûteux et pas toujours nécessaire.

De même, `PATCH` ne garantit pas forcément l'état final de la ressource : avec certains langages de "*patch*", un "*patch*", appliqué à une version différente de la ressource, peut engendrer des résultats inattendus (contrairement à `PUT`). C'est au client HTTP de s'assurer qu'il utilise bien la bonne version de la ressource. Comment ? Le plus simple est que le client demande un `Etag` (RFC 7232, section 2.3) par un `GET`, calcule les changements, avant d'envoyer le `PATCH` accompagné d'un en-tête `If-Match:` (RFC 7232, section 3.1). Ainsi, le client sera sûr de partir d'un état connu de la ressource. Cette méthode résout également le cas de deux clients tentant de "*patcher*" à peu près simultanément. (L'en-tête `If-Unmodified-Since:` est une alternative passable à `If-Match:`.) Avec certains langages de "*patch*", ou bien pour certaines opérations (par exemple ajouter une ligne à la fin d'un journal), ces précautions ne sont pas nécessaires.

En revanche, `PATCH` garantit l'atomicité. La ressource sera complètement modifiée ou pas du tout (le logiciel `patch` d'Unix ne garantit pas cela).

`PATCH` ne permet de changer que le contenu de la ressource, pas ses métadonnées. Si la requête `PATCH` a des en-têtes, ils décrivent la requête, pas la ressource et des attributs comme le type MIME ne peuvent donc **pas** être modifiés via `PATCH`.

HTTP permet de traverser des caches et ceux-ci devraient évidemment invalider une ressource pour laquelle un `PATCH` est passé.

Voici l'exemple de `PATCH` de la section 2.1, utilisant un langage de "*patch*" imaginaire, indiquée par le type MIME `application/example` :

```
PATCH /file.txt HTTP/1.1
Host: www.example.com
Content-Type: application/example
If-Match: "e0023aa4e"
Content-Length: 100

[Le patch, au format "application/example"]
```

Et son résultat (rappel : 204 indique un succès mais où la ressource n'est pas renvoyée, contrairement à 200) :

```
HTTP/1.1 204 No Content
Content-Location: /file.txt
ETag: "e0023aa4f"
```

Et si ça se passe mal ? La section 2.2 décrit les cas d'erreur possibles parmi lesquels :

— "*Patch*" invalide, inacceptable pour le langage de "*patch*" indiqué : réponse 400 (requête malformée),
— Langage de "*patch*" inconnu : réponse 415 (type inconnu),
— Requête impossible, par exemple parce qu'elle laisserait la ressource dans un état considéré par le serveur comme invalide : réponse 422,
— État de la ressource différent de ce qui est demandé, lors de l'utilisation (recommandée) de `If-Match:`, ce qui est le cas si un autre processus a modifié la ressource entre temps : 409 ou 412 selon le cas,
— Et naturellement les grands classiques de HTTP comme 403 (pas le droit de modifier cette ressource) ou 404 (pas moyen de "*patcher*" une ressource qui n'existe pas).

Pour interagir proprement avec les clients et les serveurs HTTP qui ne connaissent pas `PATCH`, la section 3 décrit l'information qu'on peut envoyer en réponse à la commande `OPTIONS`, pour indiquer qu'on accepte `PATCH` :

```
OPTIONS /example/buddies.xml HTTP/1.1
Host: www.example.com

[La réponse]

HTTP/1.1 200 OK
Allow: GET, PUT, POST, OPTIONS, HEAD, DELETE, PATCH
Accept-Patch: application/example, text/example
```

D'autre part, le nouvel en-tête `Accept-Patch:` (section 3.1 et 4.1 pour le registre IANA <`https://www.iana.org/assignments/message-headers/perm-headers.html`>) sert à préciser les formats de "*patch*" acceptés.

Comme `PATCH` modifie une ressource Web, il n'est pas étonnant que la section de ce RFC 5789 sur la sécurité soit assez détaillée. Plusieurs problèmes peuvent se poser :

— Les mêmes qu'avec `PUT` concernant l'autorisation de l'utilisateur à modifier la ressource. A priori, `PATCH` ne sera pas ouvert au public et nécessitera une authentification.
— Le risque de corruption d'une ressource (inexistant avec `PUT`) qui peut être traité par les techniques de requêtes **conditionnelles** (`If-Match:`...).
— Certains relais vérifient dans le corps de requêtes comme `PUT` ou `POST` la présence de contenus à problèmes, par exemple un virus. `PATCH` permettrait de contourner ces contrôles en envoyant le contenu en plusieurs fois. Mais le problème n'est pas très différent de celui posé par les contenus envoyés en plusieurs fois avec `Content-Range:` (le RFC cite aussi le cas des contenus comprimés, ce que je trouve moins convaincant).
— Et, comme toujours, il y a un risque de déni de service si l'application du "*patch*" peut consommer beaucoup de ressources du serveur. Celui-ci doit donc définir des limites.

Pour implémenter `PATCH`, quelques idées :

— Le livre de Lincoln D. Stein et Doug MacEachern, « "*Writing Apache modules with Perl and C*" » contient un exemple de module Perl (et de client) pour Apache,
— Pour l'instant, c'est tout ce que je vois.

Un autre article en français sur cette technologie : <`http://pierre.dureau.me/billet/2011-04-06-http-pat`