Le protocole HTTP

Olivier Aubert

Liens

- `http://www.jmarshall.com/easy/http/`
- Références: RFC1945 (HTTP1.0), RFC2616 (HTTP1.1), RFC822 (format des entêtes), RFC2396 (syntaxe des URL), RFC1521 (types MIME)
- `http://www.w3.org/Protocols/`

HTTP

- HyperText Transfer Protocol
- Au dessus de TCP/IP
- Transmission de ressources : morceau d'information identifié par une URL
- Fonctionnement client-serveur (requête/réponse)

Structure des transactions

- Messages structurés
  - Une ligne initiale
  - Zéro ou plusieurs lignes d'entête (format:
    `Header1: valeur1`
    `Header2: valeur2`
  - Une ligne vide (CRLF)
  - Le corps (optionnel) du message
- Les lignes doivent se terminer par CRLF.

Ligne initiale

- Différente suivante que le message est une requête ou une réponse
- Ligne de requête = 3 parties séparées par des espaces : nom de méthode, chemin d'accès à la ressource, version de HTTP
  `GET /~oaubert/cours/index.html HTTP/1.0`
- `GET` est la méthode la plus commune. On peut trouver aussi `HEAD`, `POST`, etc.

Ligne initiale pour la réponse

- Trois parties séparées par des espaces : version de HTTP, code de statut, description du code de statut.
  `HTTP/1.0 200 OK`
  `HTTP/1.0 404 Not Found`
- Le code est un entier sur trois chiffre dont le premier chiffre identifie la catégorie :
  - `1xx` : message d'information
  - `2xx` : succès (plusieurs variantes)
  - `3xx` : redirection
  - `4xx` : erreur du côté du client
  - `5xx` : erreur du côté du serveur
- Par exemple : `301 Moved Permanently`, `302 Moved Temporarily`, `500 Server Error`, ...

Lignes d'entête

- Ajoutent des informations à la requête ou la réponse
- Format: `Header-name: valeur` (RFC822)
- Ligne terminée par CRLF.
- Nom de l'entête non sensible à la casse (la valeur peut l'être)
- Les lignes commençant par des espaces/tabulations sont des lignes de continuation de l'entête précédent.
- HTTP1.0 (RFC1945) définit 16 entêtes, tous facultatifs
- HTTP1.1 (RFC2616) définit 46 entêtes, dont un (`Host:`) obligatoire

Exemples d'entête

- Envoyés par le client
  - `Host: www.google.com`
  - `User-Agent: Mozilla/5.0 Galeon/1.0.2 (X11; Linux i686; U;) Gecko/20011224`
  - `Accept: text/html, application/xml, image/jpeg;q=0.2`
- Envoyés par le serveur
  - `Date: Wed, 06 Feb 2002 10:25:03 GMT`
  - `Server: Apache/1.3.9 (Unix) Debian/GNU`
  - `Last-Modified: Wed, 11 Jul 2001 10:25:15 GMT`
  - `Connection: Keep-Alive`
  - `Keep-Alive: timeout=15, max=100`

Corps de message

- Le corps de message est séparé des entêtes par une ligne vide (donc deux fois CRLF)
- S'il existe un corps de message, son type et sa longueur sont précisés dans les entêtes
  - `Content-type:` indique le type MIME du document (par exemple `text/html` ou `image/jpeg`)
  - `Content-length:` indique la taille en nombre d'octets du document.

Méthode HEAD

- Utilisée pour vérifier les métadonnées d'une ressource (contenues dans les entêtes).
- Ne retourne que des entêtes, pas de corps de message

Méthode POST

- Utilisée pour envoyer des données du client vers le serveur (généralement vers un script)
- Un corps de message est présent dans la requête. Il est décrit dans les entêtes par son type et sa longueur.
- L'utilisation la plus courante de POST est pour envoyer des données depuis un formulaire. Dans ce cas :
  `Content-type:`
  `application/x-www-form-urlencoded`
  `Content-length:` longueur des données

Encodage URL

- Référence : RFC2396 (Uniform Resource Identifiers: Generic Syntax)
- Les caractères non-alphanumériques sont remplacés par `%xx` avec `xx` = code ASCII du caractère en hexadécimal.
- Les espaces sont remplacés par des `+`
- Par exemple, la chaîne `Tom & Jerry` sera transformée en `Tom+%26+Jerry`.

Évolution vers HTTP1.1

- Définition dans RFC 2616
- Améliorations apportées
  - connexions persistentes
  - meilleur contrôle des caches
  - *chunk encoding* pour renvoyer une réponse par morceaux
  - Virtual Hosting

Côté client

Les clients HTTP1.1 ont un certain nombre d'obligations :

- inclure l'entête `Host:` dans chaque requête
- accepter les réponses par morceaux (*chunks*)
- gérer les connexions persistentes, ou ajouter l'entête `Connection: close` avec chaque requête
- gérer la réponse `100 Continue`

Entête Host:

- Une adresse IP peut correspondre à plusieurs noms symboliques
- Pour pouvoir gérer cette situation au niveau des serveurs web, il faut ajouter une information concernant le nom (et éventuellement le port) du serveur interrogé.
- Une requête minimale pour un document en HTTP1.1 est donc :

```
GET /index.html HTTP/1.1
Host: www.apache.org:80
```

- Cet entête est obligatoire en HTTP1.1
- Il résout aussi certains problèmes liés aux caches

Transfert par morceaux

- Un serveur peut envoyer une réponse par morceaux (s'il ne peut pas déterminer la longueur totale par exemple)
- La version simple consiste à ajouter l'entête `Transfer-Encoding: chunked` puis à envoyer les morceaux successifs de la réponse. Après les morceaux, on envoie `0` sur une ligne, puis éventuellement des entêtes supplémentaires.
- Un morceau consiste en une ligne comportant la longueur (en hexadécimal) du morceau, suivi d'un commentaire facultatif, terminée par CRLF, puis les données terminées par CRLF

Exemple - chunk

```
HTTP/1.1 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/plain
Transfer-Encoding: chunked

1a; ignore-stuff-here
abcdefghijklmnopqrstuvwxyz
10
1234567890abcdef
0
some-footer: some-value
another-footer: another-value
[blank line here]
```


Exemple 2 - no chunk

```
HTTP/1.1 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/plain
Content-Length: 42
some-footer: some-value
another-footer: another-value

abcdefghijklmnopqrstuvwxyz1234567890abcdef
```


Connexions persistentes

- En HTTP1.0, une connexion par requête
- HTTP1.1 : connexions persistentes par défaut
- Possibilité de revenir à l'ancien fonctionnement avec l'entête `Connection: close`
- Un serveur peut fermer la connexion avant que toutes les réponses ne soient envoyées. Un client doit garder une trace des requêtes et les réenvoyer si nécessaire.
- En particulier, le client ne doit pas envoyer plusieurs requêtes s'il sait que le serveur ne supporte pas HTTP1.1

100 Continue

- Envoyé de la part d'un serveur pour indiquer au client qu'il peut continuer à envoyer des requêtes.
- Typiquement envoyé après que le client ait envoyé un début de requête avec l'entête `Expect: 100-continue`
- Utilisé dans le cas d'envoi d'un gros volume de données par le client

Côté serveur

Les serveurs HTTP1.1 ont un certain nombre d'obligations :

- exiger l'entête `Host:`
- accepter des URL absolues dans les requêtes
- accepter les requêtes par morceaux
- gérer les connexions persistentes ou inclure l'entête `Connection: close` dans chaque réponse
- envoyer la réponse `100 Continue` si besoin est
- gérer les requêtes `If-Modified-Since`
- gérer au moins les méthodes GET et HEAD
- gérer les requêtes HTTP1.0

Utilisation de Host:

- `Host:` est une mesure intérimaire. Pour l'instant, les clients qui se déclarent HTTP/1.1 et envoient une requête sans l'entête `Host` doivent recevoir une erreur `400 Bad Request`.
- Dans les versions ultérieures de HTTP, on utilisera une URL absolue (comme c'est déjà le cas pour les proxies) :
  `GET http://www.apache.org/index.html HTTP/1.1`

Requêtes par morceaux

- Les serveurs doivent être capables de comprendre des requêtes par morceaux (même si le cas est rare).

Connexions persistentes

- Si un client envoie plusieurs requêtes à travers la même connexion, le serveur doit renvoyer les réponses dans l'ordre d'arrivée des requêtes correspondantes.
- Si une requête inclut l'entête `Connection: close`, le serveur doit fermer la connexion après avoir envoyé la réponse correspondante.
- Si le serveur ne veut/peut pas implémenter de connexion persistente, il doit inclure `Connection: close` dans chaque réponse.

Datation des réponses

- Toute réponse doit inclure la date d'émission :
  `Date: Date: Fri, 31 Dec 1999 23:59:59 GMT`
- La date est précisée en GMT

Gérer If-Modified-Since

- Pour éviter de transférer des données inutilement, une requête peut être conditionnée à la date de modification du document.
- ```
  GET /index.html HTTP1.1
  If-Modified-Since:  Fri, 31 Dec 1999 23:59:59 GMT
  ```
- Pour des raisons de compatibilité, plusieurs formats de date doivent être tolérés :
  ```
  If-Modified-Since:  Fri, 31 Dec 1999 23:59:59 GMT
  If-Modified-Since:  Friday, 31-Dec-99 23:59:59 GMT
  If-Modified-Since:  Fri Dec 31 23:59:59 1999
  ```
- Seul le premier format doit être utilisé par des clients ou des serveurs HTTP1.1
- En cas de non-modification, le serveur répond :
  ```
  HTTP/1.1 304 Not Modified
  Date: Fri, 31 Dec 1999 23:59:59 GMT
  ```


Méthodes supportées

- Les serveurs HTTP1.1 doivent gérer les méthodes GET et HEAD
- D'autres méthodes existent : `PUT`, `DELETE`, `OPTIONS`, `TRACE`
- Si le serveur ne sait pas gérer une méthode, il répond :
  `HTTP/1.1 501 Not Implemented`

Compatibilité

- Pour être compatible avec les anciens clients, un serveur doit savoir gérer HTTP1.1
- En particulier, s'il reçoit une requête identifiée comme HTTP1.0 :
  - Ne pas exiger l'entête `Host:`
  - Ne jamais envoyer la réponse `100 Continue`
  ```

Range Requests

- Fonctionnalité optionnelle du protocole
- Client : envoi d'une requête avec l'entête `Range: bytes=500-600,601-999`, ou `Range: bytes=500-`
- `If-Range:` : envoi d'une partie si le document n'a pas changé, envoi de tout le document s'il a été modifié
- Serveur : envoi d'une réponse `206 Partial content`, accompagnée de l'entête `Content-Range: bytes 21010-47021/47022`

Cache - Age

- Entête `Age:` pour calculer l'âge d'un document
- Valeur : somme des temps écoulés dans chacun des caches depuis le serveur original, plus les temps de transit
- Permet de mettre en œuvre une expiration plus précise

Validateurs

- Opaques aux clients, laissés au choix des serveurs
- Validateurs forts : indiquent l'égalité des ressources
- Validateurs faibles (optionnels) : indiquent l'équivalence des ressources (pour le comptage par exemple)
- Transmis via l'entête `Etag:`
- Utilisés avec `If-Match:` et `If-None-Match:`.

Cache-Control

- Un des apports majeurs de HTTP1.1
- Catégories des directives
  - restrictions sur la cachabilité des documents (imposées par le serveur)
  - restrictions sur le droit de stocker dans les caches (imposées par le serveur ou les clients)
  - Modification des mécanismes d'expiration
  - Contrôle de la revalidation et du chargement
  - Contrôle de la transformation des ressources
  - Extensions du système de cache

Cachabilité

- Restrictions imposées par le serveur d'origine
- `public` : peut être caché et partagé
- `private` : ne peut pas être caché dans un cache public
- `no-cache` : interdiction de cacher le document
- Compatibilité HTTP/1.0 : `Pragma: no-cache`

Droit de stocker

- `no-store` : interdiction de stocker sur un support non-volatile

Mécanismes d'expiration

- imposés par le serveur d'origine ou le client
- `max-age` *`delta`* : le client accepte une réponse d'âge maximal *delta*
- `min-fresh` *`delta`* : le client veut une réponse qui sera encore valable pendant au moins *delta* secondes
- `max-stale [`*`delta`*`]` : le client accepte une réponse expirée

Revalidation

- Imposée par le client
- `no-cache` : force le rechargement depuis le serveur original
- `only-if-cached` : uniquement les documents cachés. `504 Gateway Timeout` en cas de non-présence
- `must-revalidate` : force la revalidation du document
- `proxy-revalidate` : force la revalidation sauf pour les caches privés des applications

Transformations

- Un cache peut effectuer une transformation automatique : dégradation de qualité, conversion de format
- Certaines applications ne peuvent pas tolérer de transformation (domaine médical, ...)
- `no-transform`

Warnings

- Envoyés lorsque la transparence sémantique n'est plus assurée:
- Response is stale
- Revalidation failed
- Disconnected operation
- Heuristic expiration
- Transformation applied

Nouveaux entêtes

- `Proxy-Authenticate:`, `Proxy-Authorization:` : Authentification des proxy
- `Content-MD5:` Vérification de l'intégrité
- `Content-Transfer-Encoding:` Compression des documents

Extensions futures

- Comptage du nombre d'accès (prise en compte des caches)
- Compression du protocole
- Multiplexage des flux HTTP
- Négociation transparente du contenu
- Extensibilité du protocole