Chapitre 3

Introduction à HTTP

L'*HyperText Transfer Protocol*, plus connu sous l'abréviation **HTTP** (littéralement « protocole de transfert hypertexte ») est un protocole de communication client-serveur développé pour le Web. C'est une application qui peut fonctionner sur n'importe quelle connexion fiable mais, dans les faits, on utilise le protocole TCP comme couche de transport. Un serveur HTTP utilise alors par défaut le port 80. Les clients HTTP habituels sont les navigateurs Web permettant à un utilisateur d'accéder aux données du serveur une par une. Il existe aussi des systèmes pour récuprer automatiquement tout le contenu d'un site tel que les **aspirateurs de site Web** ou les **robots d'indexation**.

3.1 HTTP 0.9

Le protocole HTTP 0.9 était extrêmement simple :

1 Connexion du client HTTP.
2 Envoi d'une requête (de méthode GET).
3 Réponse du serveur HTTP.
4 Le serveur ferme la connexion pour signaler la fin de la réponse.

La requête est de la forme :

```
GET /page.html
```

La méthode `GET` est alors la seule possible. Le serveur reconnaît de nos jours qu'il a affaire à une requête HTTP 0.9 au fait que la version n'est pas précisée suite à l'URI (contrairement à ce qui se passe pour les versions ultérieures).

La réponse est un fichier HTML :

```
<HTML>
<HEAD>
<TITLE>Exemple</TITLE>
</HEAD>
<BODY>
<P>Ceci est une page d'exemple.</P>
</BODY>
</HTML>
```

Le serveur envoie donc directement le contenu de la réponse, sans les méta-données des versions ultérieures.

3.2 HTTP 1.0

Le protocole HTTP 1.0, décrit dans la RFC 1945, prévoit l'utilisation d'en-têtes spécifiés dans la RFC 822 [RFC 822]. La gestion de la connexion reste identique à HTTP 0.9 : le client établit la connexion, envoie une requête, le serveur répond et ferme immédiatement la connexion.

3.2.1 Requête HTTP

Syntaxe.- Une requête HTTP se présente sous le format suivant :

```
Ligne de commande (Commande, URL, Version de protocole)
En-tête de requête
[Ligne vide]
Corps de requête
```

Exemple.- Voici un exemple de requête :

```
GET /page.html HTTP/1.0
Host: example.com
Referer: http://example.com/
User-Agent: CERN-LineMode/2.15 libwww/2.17b3
[Ligne vide]
```

La version du protocole HTTP est précisée suite à l'URI. L'espace n'est pas libre : il n'y a qu'un seul caractère d'espacement. La requête doit être terminée par un double retour à la ligne (`CRLFCRLF`).

Méthodes.- Outre la méthode `GET`, HTTP 1.0 supporte aussi les méthodes `HEAD` et `POST` :

- La méthode `HEAD` ne demande que des informations sur la ressource, sans demander la ressource elle-même.
- La méthode `POST` doit être utilisée pour soumettre des données en vue d'un traitement à une ressource (typiquement depuis un formulaire HTML). L'URI fournie est l'URI d'une ressource à laquelle s'appliqueront les données envoyées. Le résultat peut être la création de nouvelles ressources ou la modification de ressources existantes.

Méta-données.- On constate l'usage d'en-têtes inspirés de MIME pour transférer des méta-données :

- `Host` permet de préciser le site web concerné par la requête, ce qui est nécessaire pour un serveur hébergeant plusieurs sites à la même adresse IP (*name based virtual host*, hôte virtuel basé sur le nom). C'est le seul en-tête réellement important.
- `Referer` spécifie l'URI du document qui a donné un lien sur la ressource demandée. Cet en-tête permet aux webmaîtres d'observer d'où viennent les visiteurs.
- `User-Agent` spécifie le logiciel utilisé pour se connecter. Il s'agit généralement d'un navigateur Web ou d'un robot d'indexation.

3.2.2 Réponse HTTP

Une réponse HTTP se présente sous le format suivant :

```
Ligne de statut (Version, Code-réponse, Texte-réponse)
En-tête de réponse
[Ligne vide]
Corps de réponse
```

Exemple.- Voici un exemple de réponse :

```
HTTP/1.0 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Server: Apache/0.8.4
Content-Type: text/html
Content-Length: 59
Expires: Sat, 01 Jan 2000 00:59:59 GMT
Last-modified: Fri, 09 Aug 1996 14:21:40 GMT

<TITLE>Exemple</TITLE>
<P>Ceci est une page d'exemple.</P>
```

- La première ligne donne le **code de statut** HTTP (200 dans ce cas).
- `Date` spécifie le moment auquel le message a été généré.
- `Server` spécifie le modèle du serveur HTTP répondant à la requête.
- `Content-Length` spécifie la taille en octets de la ressource.
- `Content-Type` spécifie le type MIME de la ressource.
- `Expires` spécifie le moment après lequel la ressource devrait être considérée comme obsolète. Cela permet aux navigateurs Web de déterminer jusqu'à quand garder la ressource en mémoire cache.
- `Last-Modified` spécifie la date de dernière modification de la ressource demandée.

3.3 HTTP 1.1

Le protocole HTTP 1.1 est décrit par la RFC 2616 [HTTP 1.1] qui rend la RFC 2068 obsolète. La différence avec HTTP 1.0 est une meilleure gestion du cache. L'en-tête `Host` devient obligatoire dans les requêtes.

3.3.1 Connexions persistantes

Les soucis majeurs des deux premières versions du protocole HTTP sont, d'une part, le nombre important de connexions lors du chargement d'une page complexe (contenant beaucoup d'images

ou d'animations) et, d'autre part, le temps d'ouverture d'une connexion entre client et serveur (l'établissement d'une connexion TCP prend un temps triple de la latence entre client et serveur). Des expérimentations de connexions persistantes ont été effectuées avec HTTP 1.0 (notamment par l'emploi de l'en-tête `Connection : Keep-Alive`), mais cela n'a été définitivement mis au point qu'avec HTTP 1.1.

Par défaut, HTTP 1.1 utilise des connexions persistantes, autrement dit la connexion n'est pas immédiatement fermée après une requête, mais reste disponible pour une nouvelle requête. On appelle souvent cette fonctionnalité *keep-alive*. Il est aussi permis à un client HTTP d'envoyer plusieurs requêtes sur la même connexion sans attendre les réponses. On appelle cette fonctionnalit *pipelining*. La persistance des connexions permet d'accélérer le chargement de pages contenant plusieurs ressources, tout en diminuant la charge du réseau.

La gestion de la persistance d'une connexion est gérée par l'en-tête `Connection`, qui peut être envoyé par le client ou le serveur et contient une liste de noms spécifiant les options à utiliser avec la connexion actuelle. Si une option possède des paramètres, ceux-ci sont spécifiés par l'en-tête portant le même nom que l'option (`Keep-Alive`, par exemple, pour spécifier le nombre maximum de requêtes par connexion). Le nom `close` est réservé pour spécifier que la connexion doit être fermée après traitement de la requête en cours.

3.3.2 Négociation de contenu

HTTP 1.1 supporte la négociation de contenu : un client HTTP 1.1 peut accompagner la requête pour une ressource d'en-têtes indiquant quels sont les langues et formats de données préférés. Il s'agit des en-têtes dont le nom commence par `Accept-` :

- `Accept` liste les types MIME de contenu acceptés par le client. Le caractère étoile '*' peut servir à spécifier tous les types/sous-types.
- `Accept-Charset` spécifie les encodages de caractères acceptés.
- `Accept-Language` spécifie les langues préférées.

L'ordre de préférence de chaque option (type, encodage ou langue) est spécifié par le paramètre optionnel `q` contenant une valeur décimale entre 0 (inacceptable) et 1 (acceptable) inclus (3 décimales maximum après la virgule), valant 1 par défaut.

3.3.3 Transfert par morceaux

Le support des connexions persistantes doit également fonctionner dans les cas où la taille de la ressource n'est pas connue d'avance (comme dans le cas d'une ressource générée dynamiquement par le serveur ou un flux externe au serveur). Pour cela, l'encodage de transfert nommé *chunked* permet de transmettre la ressource par morceaux consécutifs en précédant chacun par une ligne de texte donnant la taille de celui-ci en hexadécimal. Le transfert se termine alors par un morceau de taille nulle, où des en-têtes finaux peuvent être envoyés.

Les en-têtes supplémentaires liés à cet encodage de transfert sont :

- `Transfer-Encoding` spécifie l'encodage de transfert. La seule valeur définie par la spécification RFC 2616 est `chunked`.
- `Trailer` liste tous les en-têtes figurant après le dernier morceau transféré.
- `TE` est envoyé par le client pour spécifier les encodages de contenu supportés (`Content-Encoding`, à ne pas confondre avec `Transfer-Encoding` car `chunked` est obligatoirement supporté par les clients et serveurs implémentant le standard HTTP/1.1), et spécifie si le client supporte l'en-tête `Trailer` en ajoutant `trailers` à la liste.

3.3.4 Nouvelles méthodes

Les nouvelles méthodes suivantes sont permises :

- `OPTIONS` permet d'obtenir les options de communication d'une ressource ou du serveur en général.
- `CONNECT` permet d'utiliser un proxy comme tunnel de communication.
- `TRACE` demande au serveur de retourner ce qu'il a reçu, dans le but de tester et effectuer un diagnostic sur la connexion.
- `PUT` permet de remplacer ou d'ajouter une ressource sur le serveur. L'URI fourni est celui de la ressource en question. À cause de la mauvaise implémentation des méthodes HTTP par les navigateurs, la méthode `POST` est souvent utilisée en remplacement de la requête `PUT`, qui devrait être utilisée à la fois pour la création et la mise à jour de ressources.
- `DELETE` permet de supprimer une ressource du serveur. Ces deux dernières méthodes nécessitent généralement un accès privilégié.

3.4 Exemples

Comme nous l'avons déjà dit, le client HTTP usuel est un navigateur. Sa tâche principale est d'interpréter le code HTML pour afficher ce que lui dit le code. Il est donc difficile d'observer les en-têtes avec un navigateur.

Rôle du navigateur.- Supposons que nous tapions l'adresse web :

`http://java.sun.com/index.html`

sur notre navigateur de façon à charger la page correspondante. Que se passe-t-il alors ? Le navigateur effectue les étapes suivantes :

- le navigateur observe le préfixe de l'URL (c'est-à-dire ce qui précède « `://` ») ; de « http », il déduit que le protocole que l'on veut utiliser est HTTP, aussi choisit-il le port 80 par défaut.
- il examine la partie de l'URL comprise entre la double oblique et la première oblique simple (ici « `java.sun.com` ») pour identifier l'adresse IP à laquelle se connecter. Puisque cette partie de l'URL contient des lettres, il doit s'agir d'un nom de domaine, aussi le navigateur effectue-t-il une requête auprès d'un serveur DNS pour en obtenir l'adresse IP.
- il établit alors une connexion TCP au port 80 de cette adresse IP.
- le navigateur déduit du suffixe « `/index.html` » que l'on veut voir afficher le fichier de nom physique « `/index.html` ». Il envoie donc une requête, formatée comme commande HTTP, à travers la connexion qu'il vient d'établir. Cette requête est, plus ou moins, de la forme :

  `GET /index.html HTTP/1.0`
  ligne blanche
- le serveur Web de l'ordinateur distant reçoit la requête, la décode, cherche le fichier de nom physique « `/index.html` » dont le chemin est relatif à un certain répertoire et en envoie le contenu au navigateur qui l'a demandé.
- le navigateur reçoit ce fichier, qui est censé être un fichier texte avec des en-têtes suivi d'un conteneur `HTML`, traduit le code HTML pour faire apparaître la page web. Si le fichier contient, par exemple, des images le navigateur effectue une requête `GET` par image.

Utilisation de `telnet`.- Puisqu'il est difficile d'observer les en-têtes avec un navigateur, le plus simple est d'utiliser `telnet`. Cette application permet à un utilisateur d'envoyer des caractères à un ordinateur distant et de visualiser les caractères que cet ordinateur renvoie. Lançons `telnet` en ligne de commande :

```
telnet java.sun.com 80
```

Remarquez que nous devons préciser le port car le port par défaut de `telnet` est 23, et évidemment pas 80, réservé au Web. Soyez maintenant très soigneux car vous ne pouvez pas corriger votre envoi et il n'y a pas d'écho à l'écran. Tapez :

```
GET /index.html HTTP/1.0
ligne suivante
ligne suivante
```

N'oubliez pas les deux passages à la ligne exigés par le protocole HTTP.

Le serveur envoie comme réponse le fichier (texte) demandé, qui est affiché à l'écran sans pause. On ne voit donc que la fin du fichier, dans lequel on reconnaît quelques balises HTML.

Un programme Java.- Cet exemple avec `telnet` nous montre l'extrême sensibilité de HTTP (on ne peut pas revenir en arrière lorsqu'on fait une erreur, syntaxe très rigide) et on comprend pourquoi on a intérêt à utiliser un programme, ce que nous allons faire maintenant. Utilisons, par exemple, le programme Java suivant :

```
import java.io.*;
import java.net.*;

public class WebGet
   {
   public static void main(String[] args) throws IOException
      {
      // Recuperation des arguments
      String host = args[0];
      String resource = args[1];

      // Ouverture du socket
      final int HTTP_PORT = 80;
      Socket s = new Socket(host, HTTP_PORT);

      // Obtention des flux d'octets
      InputStream in = s.getInputStream();
      OutputStream out = s.getOutputStream();

      // Obtention des flux de chaines de caracteres
      BufferedReader reader = new BufferedReader(
                              new InputStreamReader(in));
      PrintWriter writer = new PrintWriter(out);

      // Envoi de la commande
      String command = "GET /" + resource + " HTTP/1.0\n\n";
      writer.print(command);
      writer.flush();
```

```
      // Lecture de la reponse
      boolean done = false;
      while (!done)
         {
         String input = reader.readLine();
         if (input == null) done = true;
         else System.out.println(input);
         }

      // Fermeture du socket
      s.close();
      }
  }
```

Après avoir compilé le programme, commençons par le tester sur la page par défaut du serveur Apache :

```
D:\>java WebGet localhost index.html
HTTP/1.1 200 OK
Date: Mon, 09 Jan 2012 08:20:39 GMT
Server: Apache/2.2.21 (Win32)
Last-Modified: Sat, 20 Nov 2004 13:16:24 GMT
ETag: "10000000050d1-2c-3e9506e1a3a00"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<html><body><h1>It works!</h1></body></html>
```

Remarquons que, bien que nous envoyons une requête HTTP 1.0, le site renvoie une réponse HTTP 1.1. Nous savons interpréter une partie de l'en-tête : le code de statut (celui que l'on attend en général), `Date` (qui correspond bien), `Server` (évidemment la version Apache installée), `Content-Length` (qui correspond bien au nombre de caractères de la dernière ligne) et `Content-Type`. Le code HTML, très simple, se comprend bien.

Pour ne pas rester sur un exemple trivial, testons également notre programme sur le site de Google :

```
D:\>java WebGet www.google.fr index.html
HTTP/1.1 302 Found
Location: http://www.google.fr/index.html
Cache-Control: private
Content-Type: text/html; charset=UTF-8
Set-Cookie: PREF=ID=144a93e1c9104436:FF=0:TM=1325958669:LM=1325958669:
S=yXbbavo22JiF5ZBd; expires=Mon, 06-Jan-2014 17:51:09 GMT; path=/;
domain=.google.com
Date: Sat, 07 Jan 2012 17:51:09 GMT
Server: gws
Content-Length: 228
X-XSS-Protection: 1; mode=block
```

```
X-Frame-Options: SAMEORIGIN

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.fr/index.html">here</A>.
</BODY></HTML>
```

Au début du Web, la première page d'un site Web s'intitulait `index.html` ; remarquons que ceci n'est plus suivi puisque cette page n'est pas trouvée (curieusement, d'ailleurs, la page HTML censée s'afficher redonne cette URL). Comme dans le premier exemple, nous savons interpréter une partie de l'en-tête : le code de statut, `Location`, `Content-Type`, `Date`, `Server` et `Content-Length`. Le code HTML, très simple, se comprend également très bien.

3.5 Codes de statut

Voici une liste des codes de statut, ainsi que les messages généralement associés, dans l'intention de donner une courte description aux statuts représentés par ces codes. Le code numérique est destiné aux traitements automatiques par les logiciels de client HTTP. La description donne une réponse humainement compréhensible. Ces codes ont été spécifiés par la RFC 2616. Le premier chiffre du code est utilisé pour spécifier une des cinq catégories de réponse.

Code	Message	Signification
1xx	Information	
100	Continue	Attente de la suite de la requête
101	Switching Protocols	Acceptation du changement de protocole
118	Connection timed out	Délai imparti à l'opération dépassé
2xx	Succès	
200	OK	Requête traitée avec succès
201	Created	Requête traitée avec succés avec création d'un document
202	Accepted	Requête traitée mais sans garantie de résultat
203	Non-Authoritative Information	Information retournée mais générée par une source non certifiée
204	No Content	Requête traitée avec succès mais pas d'information à renvoyer
205	Reset Content	Requête traitée avec succès, la page courante peut être effacée
206	Partial Content	Une partie seulement de la requête a été transmise
3xx	Redirection	
300	Multiple Choices	L'URI demandée se rapporte à plusieurs ressources
301	Moved Permanently	Document déplacé de façon permanente
302	Found	Document déplacé de façon temporaire
303	See Other	La réponse à cette requête est ailleurs
304	Not Modified	Document non modifié depuis la dernière requête
305	Use Proxy	La requête doit être ré-adressée au proxy
307	Temporary Redirect	La requête doit être redirigée temporairement vers l'URI spécifiée

310	Too many Redirect	La requête doit être redirigée de trop nombreuses fois, ou est victime d'une boucle de redirection.
324	Empty response	Le serveur a mis fin à la connexion sans envoyer de données.
4xx	Erreur du client	
400	Bad Request	La syntaxe de la requête est erronée
401	Unauthorized	Une authentification est nécessaire pour accéder à la ressource
402	Payment Required	Paiement requis pour accéder à la ressource (non utilisé)
403	Forbidden	L'authentification est refusée. Contrairement à l'erreur 401, aucune demande d'authentification ne sera faite
404	Not Found	Ressource non trouvée
405	Method Not Allowed	Méthode de requête non autorisée
406	Not Acceptable	Toutes les réponses possibles seront refusées
407	Proxy Authentication Required	Accès à la ressource autorisé par identification avec le proxy
408	Request Time-out	Temps d'attente d'une réponse du serveur écoulé
409	Conflict	La requête ne peut pas être traitée à l'état actuel
410	Gone	La ressource est indisponible et aucune adresse de redirection n'est connue
411	Length Required	La longueur de la requête n'a pas été précisée
412	Precondition Failed	Préconditions envoyées par la requête non-vérifiées
413	Request Entity Too Large	Traitement abandonné dû à une requête trop importante
414	Request-URI Too Long	URI trop longue
415	Unsupported Media Type	Format de requête non-supporté pour une méthode et une ressource données
416	Requested range unsatisfiable	Champs d'en-tête de requête incorrect
417	Expectation failed	Comportement attendu et défini dans l'en-tête de la requêe insatisfaisable
5xx	Erreur du serveur	
500	Internal Server Error	Erreur interne du serveur
501	Not Implemented	Fonctionnalité réclamée non supportée par le serveur
502	Bad Gateway or Proxy Error	Mauvaise réponse envoyée à un serveur intermédiaire par un autre serveur
503	Service Unavailable	Service temporairement indisponible ou en maintenance
504	Gateway Time-out	Temps d'attente d'une réponse d'un serveur à un serveur intermédiaire écoulé
505	HTTP Version not supported	Version HTTP non gérée par le serveur
509	Bandwidth Limit Exceeded	Utilisé par de nombreux serveurs pour indiquer un dépassement de quota

3.6 Historique

HTTP a été inventé par Tim Berners-Lee en 1990 avec les adresses Web et le langage HTML pour créer le World Wide Web. À cette poque, le *File Transfer Protocol* (FTP) était déjà

disponible pour transférer des fichiers, mais il ne supportait pas la notion de format de données telle qu'introduite par *Multipurpose Internet Mail Extensions* (MIME). La première version de HTTP était très élémentaire, mais prévoyait déjà le support d'en-têtes MIME pour décrire les données transmises. Cette première version reste encore partiellement utilisable, connue (après coup) sous le nom de HTTP/0.9 [HTTP 0.9]. Remarquez que le numéro de port par défaut est alors 2784 et non 80.

En mai 1996, HTTP/1.0 devient finalement un standard de l'IETF et est décrit dans la RFC 1945 [HTTP 1.0]. Cette version supporte les serveurs HTTP virtuels, la gestion de cache et l'identification.

En janvier 1997, HTTP/1.1 est décrit dans la RFC 2068 de l'IETF, puis dans la RFC 2616 en juin 1999 [HTTP 1.1]. Cette version ajoute le support du transfert en pipeline et la négociation de type de contenu (format de données, langue).