

HTTP Protocole



Hyper Text Transfer Protocol

Le protocole **HTTP** est né au **CERN** en 1990.

Versions :

- **0.9** **1990** seulement GET, pas d'entête
- **1.0** **05/1996** GET, POST, HEAD entêtes
- **1.1** **01/1997** (republié en 2014)
Hébergement mutualisé
- **2.0** **2015** push, compression++,
multiplexage
- **3.0** ...

URL

Le protocole HTTP utilise les **U**niform **R**esource **L**ocator pour définir la cible des requêtes.

Où et comment

- Partie qui définit le protocole et l'adresse réseau de la ressource (serveur)

Quoi

- Partie qui définit le chemin sur le serveur

Où et comment

Protocole://nom:pass@domaine:port/

Protocole : HTTP, FTP, File etc.

Nom : identifiant utilisateur

Pass : mot de passe

Domaine : nom de domaine ou adresse IP

Port : numéro de port si différent de celui par défaut du protocole

Rem : Ces informations ne sont utilisées que par le client et ne sont pas transmises au serveur, sauf cas particulier.

Quoi

`/chemin/nom?param=val¶m...#...`

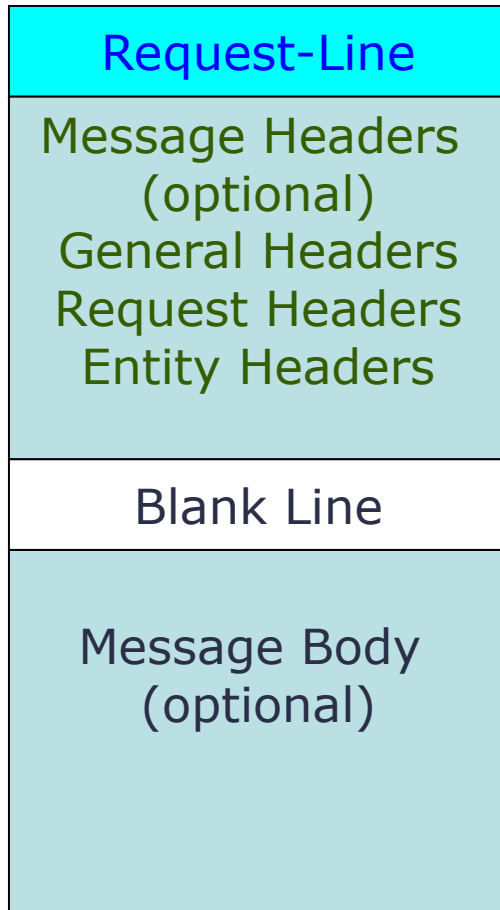
Query string : informations supplémentaires

Path : chemin d'accès à la ressource

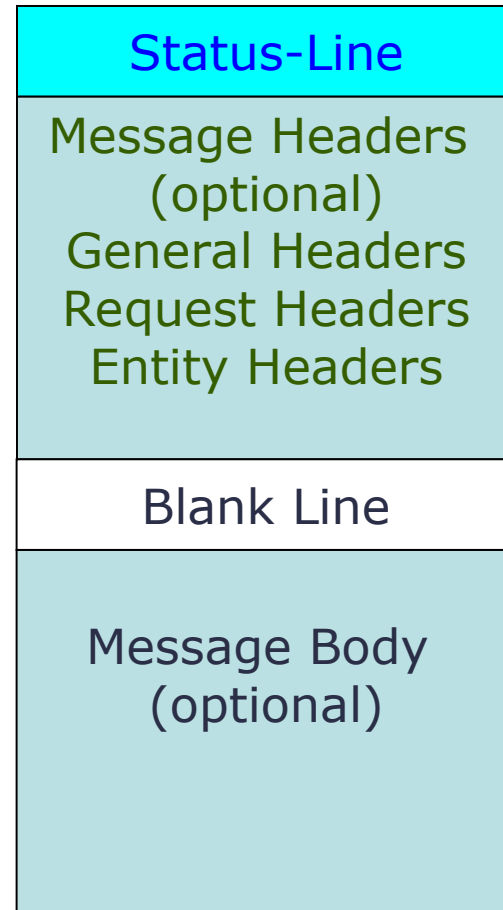
En PHP :

- `$_SERVER["REQUEST_URI"]` permet de récupérer cette information en entier
- `$_SERVER['QUERY_STRING']` ce qui est derrière le « ? », en général on passe par `$_GET` pour récupérer ces données.

Structure d'une requête et d'une réponse



Requête HTTP



Réponse HTTP

Corps (body)

Le corps contient les données transmises entre le client et le serveur. Il s'agit :

de fichiers

- html, css, images, javascript, vidéos, archives ...

Ou de contenu dynamique

- Données de formulaire, JSON, html produit par php, ...

Entête (Headers)

Les entêtes permettent des échanges d'informations complémentaires entre client et serveur afin de :

- Négocier le **contenu** : type, langue, charset ...
- Négocier la compression (quel algo)
- Gérer le **cache** du navigateur
- Authentification
- Utiliser les **cookies**
- ...

Entête (Headers)

Il y a une **centaine** de champs d'entête connus, mais il est possible d'en ajouter

- ~30 spécifiques aux **requêtes**
- ~30 spécifiques aux **réponses**
- ~40 autres usages dont certains ne sont pas encore dans la norme.

Entête (Headers)

Exemples pour une requête :

Accept: type de contenu attendu (html, css, json, image ...)

Accept-Language: langage préféré

If-Modified-Since: date de la requête précédente, ne pas renvoyer si aucune modification.

User-Agent: firefox, chrome, ...

Cookie: les cookies (r)envoyés au serveur

Authorization: information d'identification, jeton

Entête (Headers)

Exemples pour une réponse :

- Content-type:** type de contenu envoyé
(html, css, json, image ...)
- Content-language:** langue du contenu
- Last-Modified:** date du contenu
- Location:** redirige vers autre url
- Set-Cookie:** un cookie envoyé au client
- Www-authenticate:** méthodes d'authentification
permises/requises

Requête

Une requête commence toujours par une des actions suivante :

- **GET** : récupère une ressource
- **POST** : envoie donnée et récupère réponse
- **PUT** : upload de ressources
- **DELETE** : suppression de la ressource
- **CONNECT** : ouverture d'un canal de com
- **HEAD** : ne récupère pas le corps
- **OPTIONS**
- **TRACE**
- **PATCH** : modification partielle de la ressource

GET

GET /la/truc.php?toto=1&tutu=3 HTTP/1.1

Dans la requête, GET est suivi du chemin et de la « query string » et enfin de la version de HTTP utilisée.

Le corps est toujours vide.

POST

GET /la/truc.php HTTP/1.1

Dans la requête, **POST** est suivi du **chemin sans** « **query string** » et enfin de la version de HTTP utilisée.

Le **corps** contient les données envoyées.

```
POST /la/truc.php HTTP/1.1
```

```
Host: foo.example
```

```
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 27
```

```
toto=1&tutu=3
```

Réponse

HTTP/1.1 301 Moved permanently

Une réponse commence toujours la **version** du protocole suivi par un **code** et le **nom du code**

Les entêtes complètent les informations nécessaires.

Le corps contient les données s'il y en a.

Réponse

Les codes se répartissent en 5 catégories :

- 100-199 : informations avant la réponse
- 200-299 : Succès
- 300-399 : Redirection
- 400-499 : Erreur Client
- 500-599 : Erreur Serveur

À connaître par cœur

Réponse exemple

HTTP/1.1 301 Moved permanently

Vous êtes redirigés vers une autre page.

L'adresse de cette autre page est donnée dans l'entête **Location**

```
HTTP/1.1 301 Moved Permanently
Server: nginx/1.18.0 (Ubuntu)
Date: Tue, 08 Mar 2022 15:39:11 GMT
Content-Type: text/html
Content-Length: 178
Connection: keep-alive
Location: https://http.cat/301
```

Le corps est vide.

Evolution HTTP

Dans les premières versions (0.9 et 1.0)
on utilisait le principe

1 serveur = 1 site web

Pour pouvoir mettre plusieurs sites donc
serveurs sur la même machine il fallait :

- Donner plusieurs adresses IP à la machine
- Utiliser des numéros de port non standard

Evolution HTTP

La version **1.1** corrige cela et apporte une nouvelle méthode pour héberger plusieurs sites sur un même serveur.

Elle utilise le **champs** d'entête **HOST** dans les requêtes pour transmettre le nom de domaine du site contacté.

Le champ est **obligatoire** dans la requête.

→ Hébergement mutualisé

Évolution HTTP

Dans les versions 0.9 et 1.0 le client devait ouvrir une connexion pour chaque requête.

En version 1.1 plusieurs requêtes sont possibles sur la même connexion.

En version 2.0 on est plus obligé d'attendre la réponse pour envoyer la requête suivante et le serveur peut « pousser » des ressources vers le client avant qu'il ne les demande.

Proxy (Mandataire)

HTTP prévoit l'utilisation de serveur **intermédiaire** (relais ou gateway) : les **proxies**

Pour fonctionner il doivent avoir l'**URL complète**.

La requête est **modifiée** par le navigateur en ce sens.

En mettant en **cache** les réponses il permet une meilleur **performance** et diminue le trafic réseau.

Reverse Proxy

Les **reverse proxy** sont utilisés pour **rediriger** les requêtes vers plusieurs serveur fonctionnant en parallèle.

Ils permettent **d'équilibrer** la charge de travail entre ces serveur et d'améliorer la **performance** perçu du serveur.

On peut aussi les utiliser pour faire apparaître un site comme faisant partie d'un autre site de manière **transparente**