SGBLE

Zeste de savoir

HTTP 103 : les Early Hints

9 novembre 2022

Table des matières

Introduction 1
1. Le problème 1
2. Le header `Link` 2
3. Le code 103: les Early Hints 3
4. Exemple réel 3
5. Lien avec HTTP/2 Push 4
6. Support actuel (nov. 2022) 5
7. Autres ressources 6

Introduction

Ce billet présente les réponses HTTP de code `103` dites *Early Hints*, elles sont décrites dans la courte RFC 8297 de 2017, mais ce n'est que récemment qu'elles sont implémentées. Elle reste à ce jour expérimentale, par ailleurs.

1. Le problème

Sur le Web contemporain la demande d'une page (ou document) HTML implique généralement le téléchargement subséquent de plusieurs ressources accompagnantes (des *assets*): des feuilles de style, des scripts JavaScript, des polices de caractères, etc.

Ces ressources sont référencées dans le code HTML de la page demandée (ou transitivement référencées dans les références) et sont ainsi connues du navigateur dès qu'il les rencontre dans le code. Mais il est parfois intéressant de faire savoir au navigateur qu'une ressource mérite d'être préchargée avant qu'il ne la rencontre effectivement, cela afin d'améliorer la latence perçue de la part de l'audience.

À cette fin, il est possible d'utiliser l'attribut `rel="preload"` dans une balise `<link>` que l'on place général au début, dans la section `head`.

```
<!DOCTYPE html>
<html lang="en">

  <head>
    <!-- ... -->
    <!-- link de peload -->
    <link rel="preload" href="/fonts/myopenfont.woff2" as="font">
```

```
    <link rel="preload" href="/datasets/foobar.json" as="fetch">

    <!-- link classique -->
    <link rel="stylesheet" href="/styles/main.css">
    <!-- ... -->
  </head>

  <body>
  <!-- ... -->
  </body>
</html>
```

Ici est pris l'exemple d'une page Web, mais une API JSON pourrait tout autant avoir recours à une technique similaire, dans le principe.

Or, ce document HTML peut-être le résultat d'un long traitement du côté du serveur qui aura fait le travail nécessaire pour traiter la requête et produire une réponse: un appel à une base de données, des vérifications, la génération de code HTML depuis un *template*, etc. Ce temps de traitement est parfois appelé *think time*. Il ne dépend aucunement de la latence du réseau entre le client HTTP et le serveur HTTP.

Pendant ce *think time* qui n'induit pas de charge réseau, il pourrait être intéressant de faire savoir au navigateur les futures ressources à précharger. Mais cela est impossible en les référençant dans le document HTML précisément en attente de rendu.

2. Le header `Link`

Référencer les indices de préchargement dans le corps de la réponse a donc ses limites. Mais il est possible de les référencer directement au niveau du protocole HTTP, avec le *header* `Link` qui se présente comme suit (en HTTP/2, les *headers* ne sont pas capitalisés):

```
HTTP/2 200 OK
link: </fonts/myopenfont.woff2>; rel=preload; as=font
link: </datasets/foobar.json>; rel=preload; as=fetch
...
...
```

Le serveur pourrait techniquement envoyer les headers de façon prématurée avant le corps de la réponse, à savoir le document HTML. Hélas dans la pratique ce n'est pas aussi simple: les *headers* peuvent être eux aussi conditionnés par le traitement de la requête, tout comme le corps. Connaître la bonne valeur d'un *header* `Content-Length`, ou bien savoir s'il faut renvoyer une réponse en succès ou en erreur en premier lieu requiert de passer par le traitement complet de la requête dans la plupart des cas. On n'échappe donc pas au *think time*.

La RFC 8297 reconnaissait cet état de choses, motivant ainsi l'idée des *Early Hints*.

3. Le code 103 : les Early Hints

Pour contourner cette limitation, on peut renvoyer une réponse prévisionnelle contenant les *headers* `Link`, mais sans la charge utile qui n'existe pas encore. C'est une réponse en code `103 Early Hints` qui remplit cet office.

Elle est ensuite suivie de la réponse définitive (en statut `200 OK` par exemple) contenant le corps de la réponse après que le serveur eut fini son traitement.

```
HTTP/2 103
link: </fonts/myopenfont.woff2>; rel=preload; as=font
link: </datasets/foobar.json>; rel=preload; as=fetch
```

(attente….)

```
HTTP/2 200
date: Tue, 11 Oct 2022 21:25:28 GMT
content-type: text/html;charset=UTF-8
content-length: 326
link: </fonts/myopenfont.woff2>; rel=preload; as=font
link: </datasets/foobar.json>; rel=preload; as=fetch
...
...
```

> En général, la réponse définitive contient elle aussi les mêmes *headers* `Link` que la réponse prévisionnelle qui précède.

Le serveur HTTP supportant la fonctionnalité renvoie ainsi une telle réponse `103` au client qui a donc connaissance des éléments qu'il *pourrait* télécharger en attendant une réponse définitive.

4. Exemple réel

J'ai mis en place un serveur minimal qui sert deux *endpoints*:

— `/`: un document HTML avec 2 secondes de temps de traitement;
— `/style.css`: un document CSS statique avec un temps de traitement instantané.

Avec `curl`, voici ce que l'on peut obtenir (ce qui est préfixe d'un `>` vient du client, ce qui est préfixé par `<` vient du serveur):

```
% curl -v https://test-103.sagebl.eu/
[...]

> GET / HTTP/2
> Host: test-103.sagebl.eu
> user-agent: curl/7.82.0
> accept: */*
[...]

< HTTP/2 103
< link: </style.css>; as=style; rel=preload


[approximativement 2 secondes d'attente ici]


< HTTP/2 200
< date: Tue, 11 Oct 2022 21:25:28 GMT
< content-type: text/html;charset=UTF-8
< content-length: 326
< link: </style.css>; rel="preload"; as="style"
<!DOCTYPE html>
    <html lang="en">
      <head>
        <meta charset="utf-8">
        <link rel="stylesheet" href="/style.css" />
        <title>Home</title>
      </head>
      <body>
        <h1>Hello, World</h1>
[...]
[...]
```

Bien que `curl` ne précharge pas `/style.css` (ce n'est pas un navigateur), il reçoit bien la réponse `103` bien avant de recevoir la réponse définitive `200` approximativement 2 secondes plus tard.

i

Ce serveur utilise Cloudflare qui a une heuristique pour déterminer si le client est digne de recevoir une pré-réponse `103` ou non. Il faut parfois faire plusieurs tentatives pour obtenir le résultat montré ci-dessus.

5. Lien avec HTTP/2 Push

Un navigateur supportant les *Early Hints* n'est en rien obligé de précharger quoi que ce soit après avoir pris connaissance de la réponse `103`. Il peut estimer que le préchargement n'est pas

pertinent selon ses propres critères. Il s'agit bien d'indices donnés à titre indicatif (*hint*), mais pas d'instructions impératives.

Cette approche distingue les *Early Hints* d'une autre fonctionnalité qui avait initialement été introduite avec HTTP/2: le *Server Push* . Cette ancienne méthode consiste à envoyer de façon spéculative des ressources jugées «utiles» en même temps que la page initialement demandée. L'expérience semble montrer que cette méthode peut faire plus de mal que de bien, en forçant le téléchargement de ressources finalement inutiles, gaspillant ainsi de la bande passante.

La fonctionnalité *Server Push* est aujourd'hui dépréciée et désactivée dans certaines implémentations, dont Google Chrome 106 . Sous Firefox, vous pouvez d'ores et déjà désactiver cela en fixant `network.http.http2.allow-push=false`.

6. Support actuel (nov. 2022)

Dans ce billet, les exemples ont tous été donnés avec HTTP/2. Cependant, il est à noter que la RFC 8297 n'exclut en rien la version HTTP/1.1 (les exemples donnés dans la RFC l'utilisent). Mais il semblerait que seules les implémentation de HTTP/2 et HTTP/3 prendront en charge les réponses `103` en pratique. Il n'est d'ailleurs pas garanti que toutes les implémentations clientes de HTTP/1.1 supportent de recevoir plusieurs réponses subséquentes, ce qui est nécessaire pour recevoir une réponse code `103` (voire plusieurs) suivie d'une réponse ultime (par exemple en code `200`).

Ainsi, voici le support disponible à l'heure actuelle:

Support côté CDN et serveurs:

- Cloudflare pour HTTP/2 et HTTP/3 seulement (à activer explicitement),
- Fastly pour HTTP/2,
- NGINX: en cours d'étude et de développement
- Apache 2.4 : `H2EarlyHints on`
- HAProxy 1.9 : `http-request early-hint Link "</style.css>; rel=preload; as=style"`

Support côté client:

- Chrome depuis la version 103 (coïncidence ?) pour HTTP/2 et HTTP/3 seulement,
- Firefox avec `network.early-hints.enabled=true` pour le moment (sur la version 107, je n'ai pas eu l'occasion de voir à quel point ça fonctionne),
- `curl`, comme toute implémentation HTTP/2 correcte, s'accommode du fait d'obtenir une (ou des) réponses en code `103` suivies d'une réponse définitive (ex.: en code `200`) comme vu, bien qu'aucun préchargement effectif n'a lieu.

De façon générale, les clients HTTP/2 et HTTP/3 ne devraient pas avoir de problème avec les réponses multiples subséquentes, même si le support d'une réponse en code `103` n'est pas spécifiquement prévu dans l'implémentation (sans doute sera-t-elle simplement ignorée).

7. Autres ressources

Quelques articles de grand acteurs, par ordre chronologique inversé. On voit que le sujet est abordé depuis longtemps, même si les mise en œuvre sont récentes.

— https://developer.chrome.com/blog/early-hints/ (juin 2022)
— https://blog.cloudflare.com/early-hints/ (sept. 2021)
— https://www.fastly.com/blog/faster-websites-early-priority-hints (juin 2018)