1. Http2 Client

1.1. Préambule sur le protocole HTTP/2 :

La version **1.1** du protocole Http date de **1997**.

La version 2 du protocole Http a été standardisée en **2015** (en s'inspirant du protocole expérimental SPDY de google commencé en 2009)

La plupart des navigateurs web sont aujourd'hui compatible avec **http/2** .

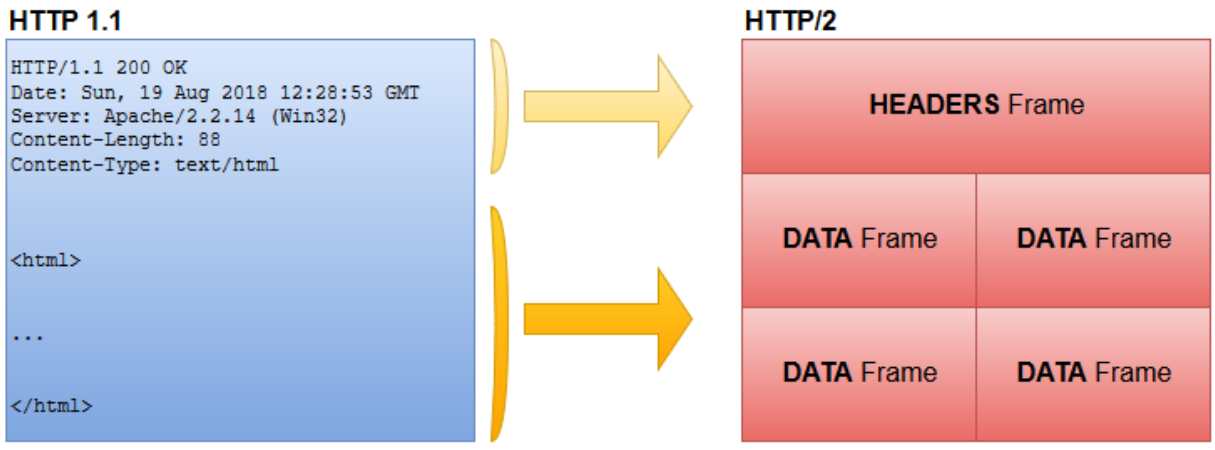
HTTP/2 conserve une rétrocompatibilité complète avec HTTP 1.1 . Mêmes syntaxes .

http/1 échange la plupart des données en mode texte.

http/2 échange (en interne) les données en mode binaire (même si du coté "client" et "serveur" une vision "mode texte" est toujours de mise pour les types MIMES "text/html" , ...)

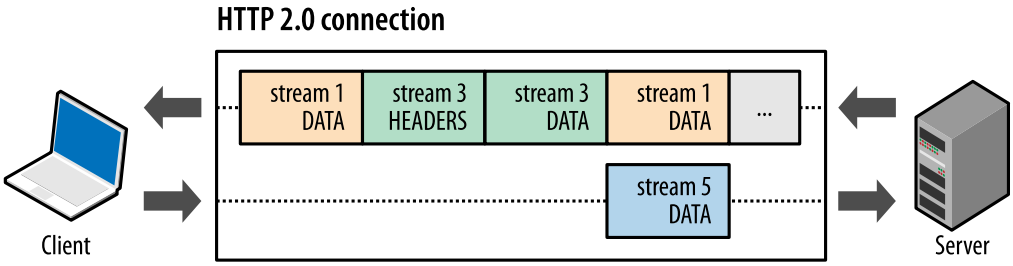
Une connexion http/1 ne pouvait récupérer qu'une seule ressource à la fois . Malgré la réutilisation de connexion (keep-alive) , et l'éventuelle parallélisation des connexions tcp/ip vers un même domaine (limitées à 6) , un chargement d'un grand nombre de ressources via http est plutôt séquentiel en http/1 et peut donc prendre beaucoup de temps.

Une connexion **http/2** peut être utilisée en mode "**multiplexage de flux**" et on peut ainsi récupérer plusieurs ressources en parallèle . Ceci permet de beaucoup améliorer la vitesse de certains téléchargement .



En interne, le protocole **http/2** véhicule des données dans des blocs binaires appelées "**Frames**" (avec parties "***header***" et "***data***") .

Plusieurs "frames" peuvent ainsi transitées (éventuellement dans les 2 sens en mode "bi-directionnel" au sein d'une même connexion http/2 . Les "frames" peuvent éventuellement être priorisées .



NB : **au sein d'une même connection HTTP2 établie, le client et le serveur peuvent échanger** (dans les 2 sens) **des frames** (HEADERS + DATA) **comme s'il s'agissait de requêtes indépendantes** .
Chaque frame (HEADERS ou DATA) **comporte un attribut** important **.streamId .**

Ceci permet au client de comprendre qu'une frame de réponse est associée à telle frame de requête (par analyse de **corrélation** du **stream ID**) .

Le protocole **http/2** autorise également un mode **push** (où le serveur peut prendre l'initiative de renvoyer spontanément certaines pièces attachées par encore demandées) .

Le protocole **http/2** est capable d'effectuer automatiquement **une compression** (en binaire optimisé) **des entêtes HTTP** ce qui permet d'**optimiser la bande passante .**

Pour approfondir --> *https://oxiane.developpez.com/tutoriels/java/http2-java/*

1.2. Présentation de l'api java HttpClient

La nouvelle API **HTTP Client** (en pré-version "incubator" en v9 puis standard depuis v 11) propose **un support des versions 1.1 et 2 du protocole HTTP** ainsi que les WebSockets côté client.

Cette api (beaucoup plus moderne que l'ancienne classe HttpURLConnection datant du jdk 1.1) **peut fonctionner en mode asynchrone** et permet (si nécessaire) l'utilisation de l'API Flow (reactive streams) pour fournir les données du body d'une requête (Flow.Publisher) et consommer le body d'une réponse (Flow.Subscriber).

L'API est contenue dans le package **java.net.http** du module java.net.http. Elle contient plusieurs types dont les principaux sont ***HttpClient*** , ***HttpClient.Builder*** , ***HttpRequest*** , ***HttpRequest.Builder*** , ***HttpRequest.BodyPublisher*** , ***HttpRequest.BodyPublishers*** , ***HttpResponse*** , HttpResponse.ResponseInfo , ***HttpResponse.BodySubscriber*** , ***HttpResponse.BodySubscribers*** , HttpResponse.BodyHandler , HttpResponse.BodyHandlers .

Généralités : Un ***XxxBuilder*** sert à construire une instance immuable de type ***Xxx***

Les XxxYyyys sont des fabriques de XxxYyy

Un "*Handler*" sert à décortiquer la réponse (et la récupérer dans un certain type) en s'aidant des informations d'entêtes trouvées dans HttpResponse.ResponseInfo

1.3. Exemples avec api HttpClient

Exemple en mode synchrone :

```
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.net.http.HttpResponse.BodyHandlers;
...
  public static void test_new_http2_client_since_java9_standard_since_java11() {
        HttpClient client = HttpClient.newHttpClient();

        HttpRequest req =
                HttpRequest.newBuilder(URI.create("http://www.google.com"))
                        .header("User-Agent","Java")
                        .GET()
                        .build();

     try {
        HttpResponse<String> resp = client.send(req, BodyHandlers.ofString());
        System.out.println("reponse status:" + resp.statusCode()); //200 for ok
        System.out.println("reponse uri:" + resp.uri().toString()); //http://www.google.com
        System.out.println("reponse type:" + resp.headers().map().get("Content-Type"));
                        //--> [text/html; charset=ISO-8859-1]
        System.out.println("reponse text:" + resp.body());//texte html
        } catch (IOException e) {
                e.printStackTrace();
        } catch (InterruptedException e) {
                e.printStackTrace();
        }
  }
```

```
HttpClient client = HttpClient.newHttpClient(); //Http2 par défaut si possible
```

ou bien

```
HttpClient httpClient =  HttpClient.newBuilder()
       .version(HttpClient.Version.HTTP_1_1)
       //.proxy(ProxySelector.of(new InetSocketAddress("proxy.xyz.com", 80)))
       .connectTimeout(Duration.ofSeconds(10))
       .build();
```

Préparation d'une requête à envoyer en mode POST :

```
HttpRequest requetePost = HttpRequest.newBuilder()
```

```
.uri(URI.create("https://www.xyz.com/api-xyz"))
.setHeader("Content-Type", "application/json")
.POST(BodyPublishers.ofString(dataAsJsonString))
.build();
```

Quelques Variantes :

```
.POST(HttpRequest.BodyProcessor.fromByteArray(sampleData))
```

ou encore

```
.POST(HttpRequest.BodyProcessor.fromFile(
      Paths.get("src/test/resources/sample.json")))
```

Variante du premier exemple en **mode asynchrone** :

```
..
//En mode asynchrone , sendAsync retournant un CompletableFuture :
client.sendAsync(req, BodyHandlers.ofString())
        .thenAccept(resp -> {
                System.out.println("recuperation réponse asynchrone / interpreted by "
                                + Thread.currentThread().getName());
                System.out.println("reponse status:" + resp.statusCode());
                System.out.println("reponse uri:" + resp.uri().toString());
                System.out.println("reponse type:" + resp.headers().map().get("Content-Type"));
                System.out.println("reponse text size:" + resp.body().length());
                });

 System.out.println("suite synchrone interpreted by " + Thread.currentThread().getName());
        try { Thread.sleep(2000);//pause ici pour eviter arrêt complet du programme
                                // avant la fin des taches de fond asynchrones
        } catch (InterruptedException e) {          e.printStackTrace();      }
System.out.println("fin synchrone / interpreted by " + Thread.currentThread().getName());
```

Résultats :

```
suite synchrone interpreted by main
recuperation réponse asynchrone / interpreted by ForkJoinPool.commonPool-worker-3
reponse status:200
reponse uri:http://www.google.com
reponse type:[text/html; charset=ISO-8859-1]
reponse text size:12791
fin synchrone / interpreted by main
```

Avec subscriber/reactive stream (pour gérer réponse) :

```
public static void test_new_httpClient_withSubscriber() {
        HttpClient client = HttpClient.newHttpClient();
        HttpRequest req =
                HttpRequest.newBuilder(URI.create("http://www.google.com"))
                        .header("User-Agent","Java").GET().build();
        MySimplePrintSubscriber subscriber1 = new MySimplePrintSubscriber();
        //NB ;  MySimplePrintSubscriber est une implémentation simple de Flow.Subscriber<Object>

        // BodyHandlers.fromLineSubscriber(s) for registering a subscriber that will receive response text line by line
        // the return CompletableFuture contains no body but status and other infos
    client.sendAsync(req, BodyHandlers.fromLineSubscriber(subscriber1))
            .thenApply(HttpResponse::statusCode) //extract statusCode from httpResponse
            .thenAccept((status) -> {
               if (status != 200) { System.err.printf("ERROR: %d status %n", status); }
            });
    System.out.println("suite synchrone interpreted by " + Thread.currentThread().getName());
        try {
        Thread.sleep(2000);//pause ici pour éviter arrêt complet du programme
                                // avant la fin des taches de fond asynchrones
        } catch (InterruptedException e) {e.printStackTrace();}
    System.out.println("fin synchrone / interpreted by " + Thread.currentThread().getName());
}
```

Résultat de type :

```
mySimplePrintSubscriber>> Received onNext(item) with item=ligne1
mySimplePrintSubscriber>> Received onNext(item) with item=ligne2
mySimplePrintSubscriber>> Received onNext(item) with item=ligne3
```