# 一名前端眼中的GRPC

作者: 郭政鸿 2021/4/15

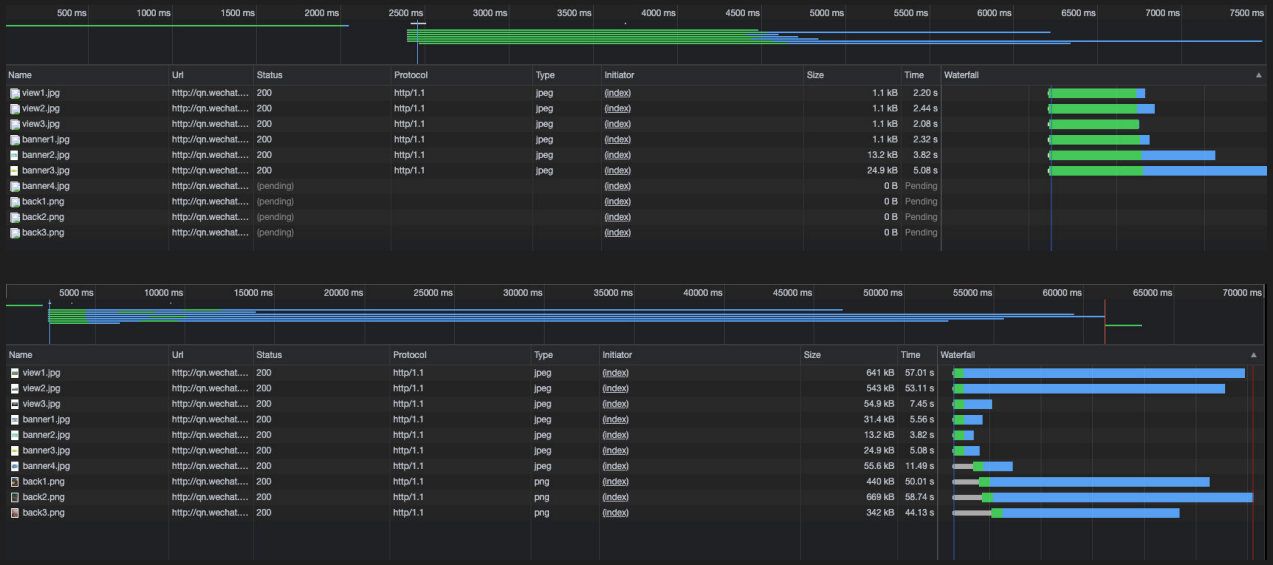
## 1、GRPC前置知识

grpc是一款由google基于HTTP/2开源的一款rpc框架, 通过ProtoBuf序列化, 支持多语言. 由于是基于HTTP/2, 所有在开始GRPC之前, 我们需要先了解的什么是HTTP/2, 它为什么会出现, 它的出现解决了什么问题.

我们目前用的最多的是HTTP/1.1它存在着这些问题

- 同一个域名下连接数限制
- 线头阻塞问题(按照FIFO规则处理)
- 明文传输不安全
- HTTP头部信息大量重复

### 1) 连接数限制



通过测试发现 `chrome 89` 支持的同一域名最大连接数是为6, 浏览器对于同一域名的最大同时连接数是有限制的, 具体限制数量不同浏览器的值不一样.

## 2) 线头阻塞问题

浏览器内每个TCP连接只能按照FIFO的规则同时处理一个请求并作出响应, 如果上一个请求没有做出响应, 后续的 `请求` 和 `响应` 都会被阻塞. 为了解决此问题出现了 `管线化` 技术, 但是并没有完全解决这个问题, 如果前面的请求没有响应, 后续的 `响应` 还是会被阻塞. 根本问题

**响应还是按顺序返回的**

并且如果使用代理等, 要求 `客户端` 、 `代理` 、 `服务器` 都要支持管线化

## 3) header头部重复

每次请求会存在大量重复的头部信息, 造成资源浪费.

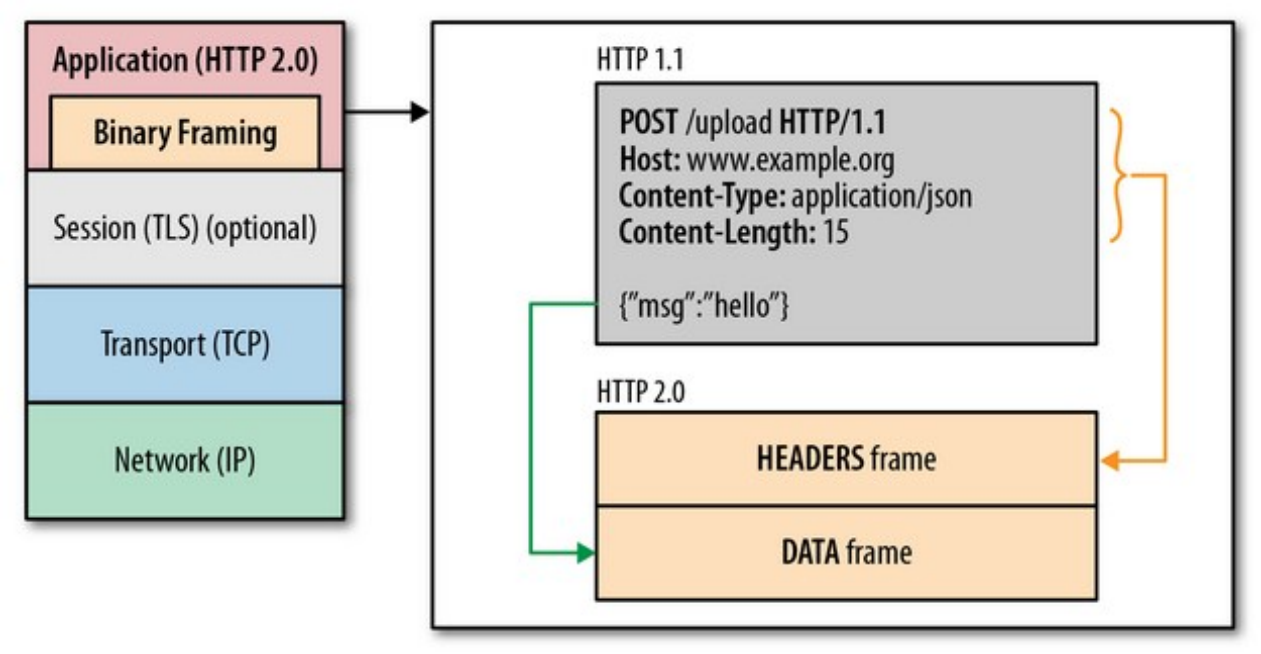
## 4) 明文传输不安全

grpc采用protobuf进行序列化成二进制数据, 而protobuf数据本身是无 `自描述` 的, 所以数据相对安全, HTTP/2虽然没有强制要求使用TSL, 但是一般都用, 越早用越好!

## 5) HTTP/2是如何解决上述问题的

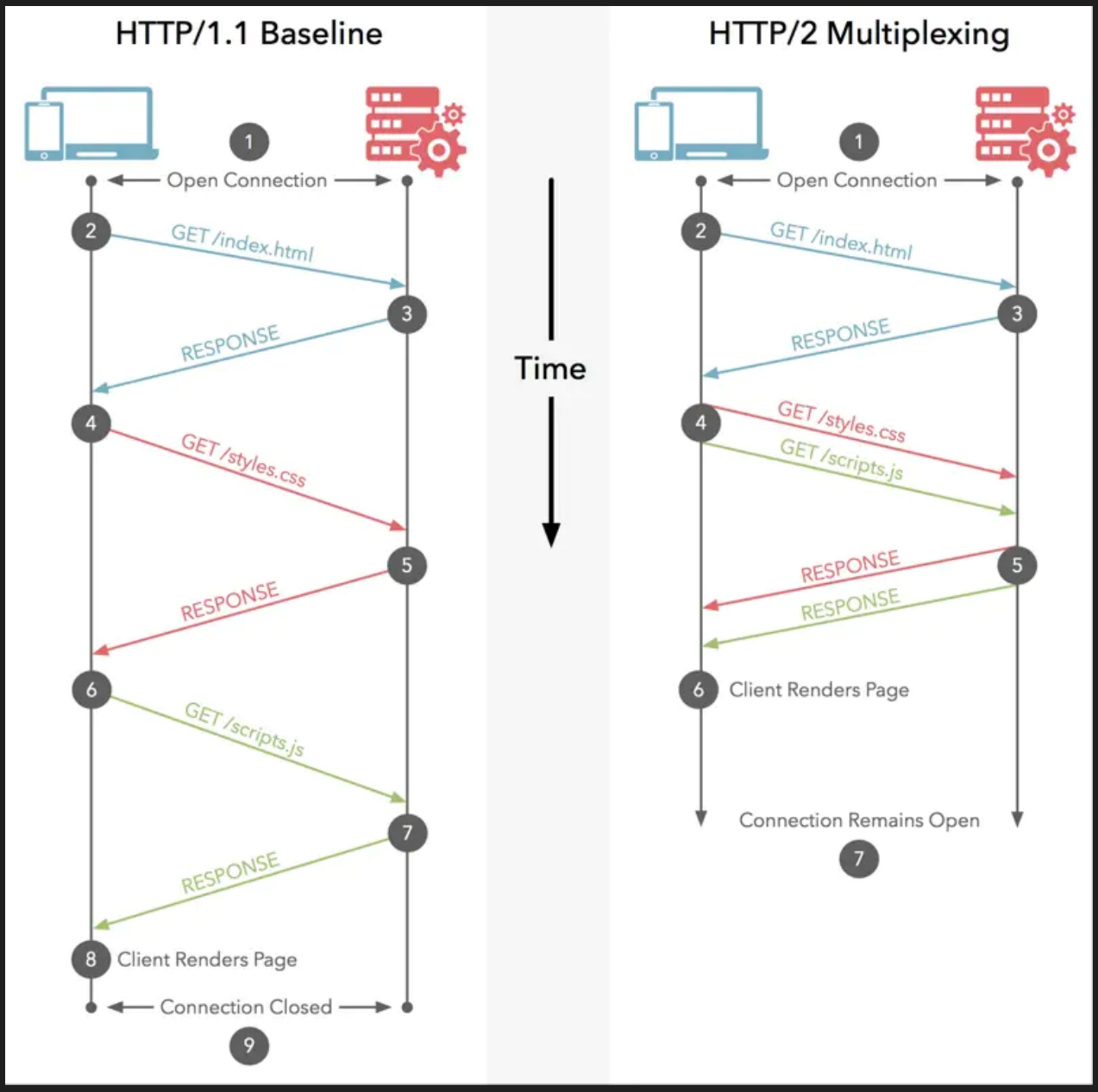
① 二进制分帧层

HTTP/2引入了二进制分帧层, 帧是数据传输的最小单位, 以二进制代替明文传出, 然后HTTP/2通信都在一个连接上完成，这个连接可以承载任意数量的双向数据流。相应地，每个数据流以消息的形式发送，而消息由一或多个帧组成，这些帧可以乱序发送，然后再根据每个帧首部的流标识符重新组装。




② 多路复用

客户端可以向对方不断的发送帧, 通过每个帧的 `stream identifier` 来标明每一帧数据哪个流(流可以类比为HTTP/1.1中的每个请求), 请求响应分成多个帧, 在不同的流中, 帧可以交错的进行传输. 这就是HTTP/2的多路复用




流的概念实现了单连接上多个请求-响应的并行, HTTP/2对于同一的域名只需要创建一个连接, 减少了建立TCP连接所带来的慢启动问题

当然HTTP/1.1中我们可以有一定的优化, 不影响首屏渲染的样式或者脚本可以通过 `defer` 、 `async` 进行延迟加载, 我们可以通过TCP预热来进行进行预加载等...

TCP预热: `<link rel="prefetch" src='xxxx' />` 或者preload等

③ 服务端推送

客户端发送一个请求, 建立连接后, 服务器主动向浏览器推送与这个请求相关的资, 这样浏览器就不需要发起后续请求了, 在HTTP/1.1中有 websocket、SSE(server-sent events)

④ 设置优先级

HTTP/2里可以为每个stream设置 `依赖` 和 `权重` , 可以按照依赖树和优先级来解决线头阻塞问题.

# 2、protobuf

开头的时候我们说了grpc是基于HTTP/2和protobuf, 简单的介绍了HTTP/2, 下面是protobuf

grpc序列化支持 PB（Protocol Buffer）和 JSON， PB 是一种语言无关的高性能序列化方案, 它规定了数据序列化的规则.

PB的优势和劣势

**优势:**

- 传输数据更小
- 序列化和反序列化更快
- 由于传输的过程中使用的是二进制，没有结构描述文件，无法解析内容，安全性更高

**劣势:**

- 由于传输过程使用的是二进制，自解释性较差，需要原有的结构描述文件才能解析

**实际数据对比:**

- 序列化速度：比JSON快20-100倍
- 数据大小：序列化后体积小3倍

基本的protobuf文件, 现在大部分使用的protobuf3版本

```
syntax = "proto3";

package chat.room;

enum UserType {
  TEACHE = 0;
  STUDENT = 1;
}

message UserInfo {
  string userName = 1;
  string avatar = 2;
  string recentMessage = 3;
  string recentTime = 4;
  UserType userType = 5;
}

message GetUserInfoResponse {
  repeated UserInfo userInfoList = 1;
}

message GetUserInfoRequest {}

service ChatRoomService {
  rpc getUserInfo (GetUserInfoRequest) returns (GetUserInfoResponse);
}
```

注意点: 枚举类型中的第一个数值必须为0, 其他的一般从1开始

使用nodeJS编写客户端和服务端demo代码~

在浏览器环境使用grpc需要以下几步

1. 使用protoc编译工具 和 grpc-web插件将 PB 文件编译

2. 服务端实现GRPC接口
3. 配置NGINX/envoy代理, 进行协议转换
4. webpack打包CommonJS代码

## 编译protobuf文件为 `commonjs` 规范js

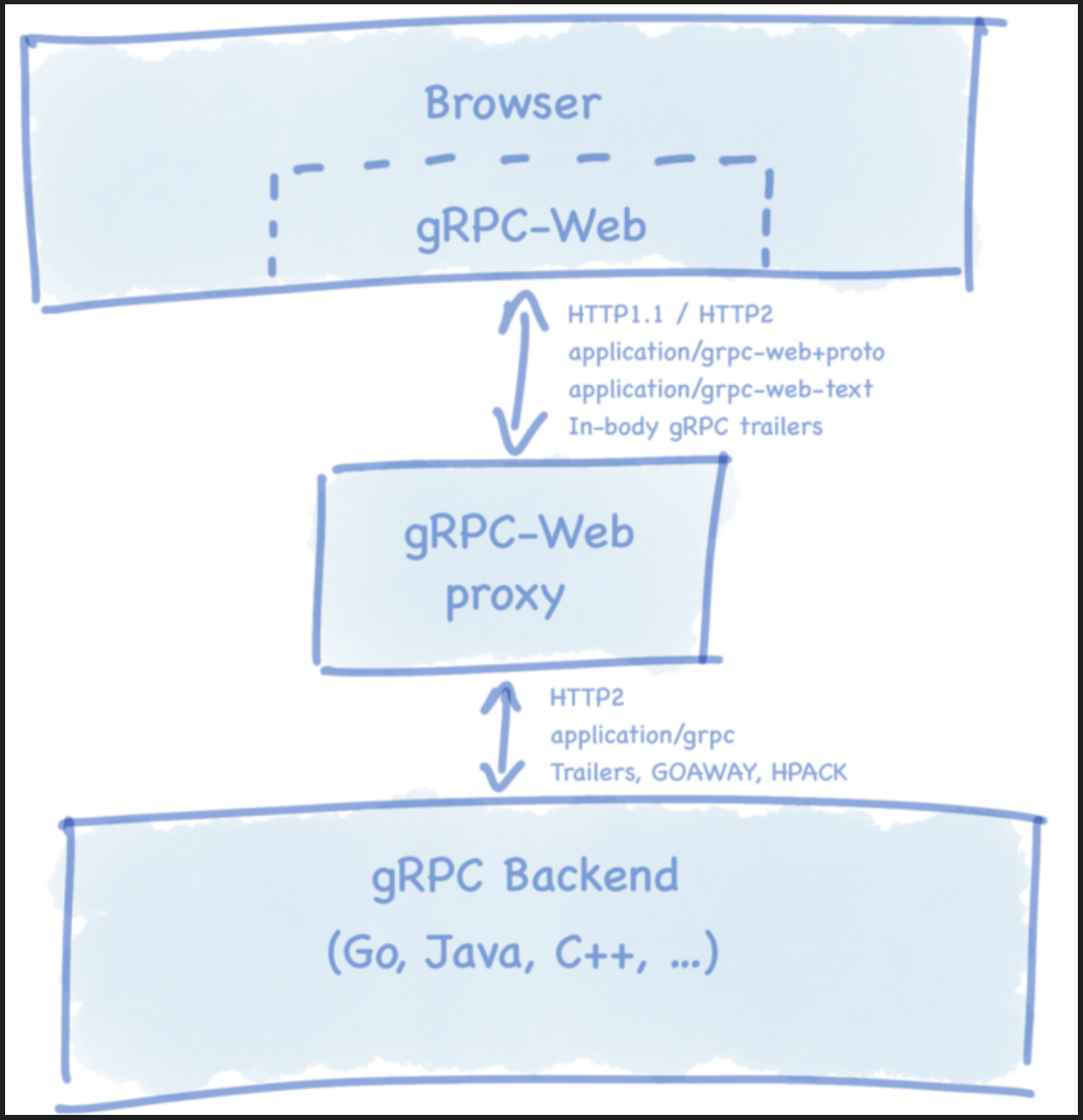
```
protoc ./chat.proto --js_out=import_style=commonjs:./ --grpc-web_out=import_style=commonjs,mode=grpcweb:./
```

> 注意: 此处mode为grpcweb而不是grpcwebtext, 与content-type相关

通过上述命令我们可以得到两个文件

- chat_grpc_web_pb.js -- 封装了rpc接口的请求类
- chat_pb.js -- 封装了数据约束的类, 可类比为typescript的types

## 配置grpc代理




首先从一个简单的nginx代理开始

```
server {
  listen       80;
  server_name  www.guetweb.com;

  location / {
    proxy_pass http://localhost:8080;
  }
}
```

使用HTTP/2

```
server {
  listen       80 http2;
  server_name  www.guetweb.com;

  location / {
    proxy_pass http://localhost:8080;
  }
}
```

完整的grpc代理配置, 支持跨域

```
server {
    listen       443 ssl http2;
    server_name  www.guetweb.com;

    # ssl证书地址
    ssl_certificate     /usr/local/etc/nginx/cert/guetweb.com/www.guetweb.com.crt;  # pem文件的路径
    ssl_certificate_key  /usr/local/etc/nginx/cert/guetweb.com/www.guetweb.com.key; # key文件的路径

    # ssl验证相关配置
    ssl_session_timeout  5m;    #缓存有效期
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE:ECDH:AES:HIGH:!NULL:!aNULL:!MD5:!ADH:!RC4;    #加密算法
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;    #安全链接可选的加密协议
    ssl_prefer_server_ciphers on;   #使用服务器端的首选算法

    location / {
        grpc_set_header     Content-Type application/grpc;




        grpc_pass           grpc://localhost:2022;




        if ($request_method = 'OPTIONS') {
            add_header 'Access-Control-Allow-Origin' "*";
            add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS, DELETE';
            add_header 'Access-Control-Allow-Headers' 'DNT,X-CustomHeader,Keep-Alive,User-Agent,X-
Requested-With,If-Modified-Since,Cache-Control,Content-Type,Content-Transfer-Encoding,Custom-Header-
1,X-Accept-Content-Transfer-Encoding,X-Accept-Response-Streaming,X-User-Agent,X-Grpc-Web';
            add_header 'Access-Control-Max-Age' 1728000;
            add_header 'Access-Control-Expose-Headers' 'Content-Transfer-Encoding';
            add_header 'Content-Type' 'text/plain charset=UTF-8';
            add_header 'Content-Length' 0;
            return 204;
        }

        if ($request_method = 'POST') {
            add_header 'Access-Control-Allow-Origin' '*';
            add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';
            add_header 'Access-Control-Allow-Headers' 'DNT,X-CustomHeader,Keep-Alive,User-Agent,X-
Requested-With,If-Modified-Since,Cache-Control,Content-Type,Content-Transfer-Encoding,Custom-Header-
1,X-Accept-Content-Transfer-Encoding,X-Accept-Response-Streaming,X-User-Agent,X-Grpc-Web';
            add_header 'Access-Control-Expose-Headers' 'Content-Transfer-Encoding,Grpc-Message,Grpc-
Status';
        }
    }
}
```

参考地址:

https://www.grpc.io/

https://developers.google.com/protocol-buffers

https://www.nginx.com/blog/nginx-1-13-10-grpc

https://github.com/grpc/grpc-web

> https://github.com/grpc/grpc-node/tree/master/packages/grpc-js
>
> https://github.com/protobufjs/protobuf.js
>
> https://github.com/grpc-ecosystem/grpc-gateway