2022-09-07

1，整理http协议http2协议http3协议

一、HTTP

HTTP协议(超文本传输协议HyperText Transfer Protocol)，它是基于TCP协议的应用层传输协议，简单来说就是客户端和服务端进行数据传输的一种规则。

HTTP是一种**无状态** (stateless) 协议, HTTP协议本身不会对发送过的请求和相应的通信状态进行持久化处理。这样做的目的是为了保持HTTP协议的简单性，从而能够快速处理大量的事务, 提高效率。

然而，在许多应用场景中，我们需要保持用户登录的状态或记录用户购物车中的商品。由于HTTP是无状态协议，所以必须引入一些技术来记录管理状态，例如Cookie。

1.HTTP的工作流程

浏览器与服务器建立TCP连接，即三次握手

TCP连接成功，浏览器发出HTTP请求命令

服务端接收请求并返回HTTP响应

服务器关闭连接，即四次挥手

浏览器解析请求的资源

HTTP报文

2.分为请求和响应报文

请求行：包含请求方法，请求的URI（统一资源标识符）和HTTP版本

状态行：包含响应结果的状态码

首部字段：包含请求和响应的各种条件、属性的各类首部

其他：包含未定义的首部如Cookie等

connection：keep-alive可以让TCP连接保持打开，浏览器可通过相同的连接发送请求

队头阻塞是指：当顺序发送的请求序列中的一个请求因为某种原因被阻塞时，在后面排队的所有请求也一并被阻塞，会导致客户端迟迟收不到数据。

3.解决队头阻塞

将同一页面的资源分散到不同域名下 ，提示连接上限。Chrome，对于同一个域名允许同时建立6个TCP持久连接，使用TCP连接时，虽然能共用一个TCP管道，但在一个管道中同一时刻只能处理一个请求。

Spriting合并多张小图2位一张大图，再用JavaScript或Css将小图重新分割

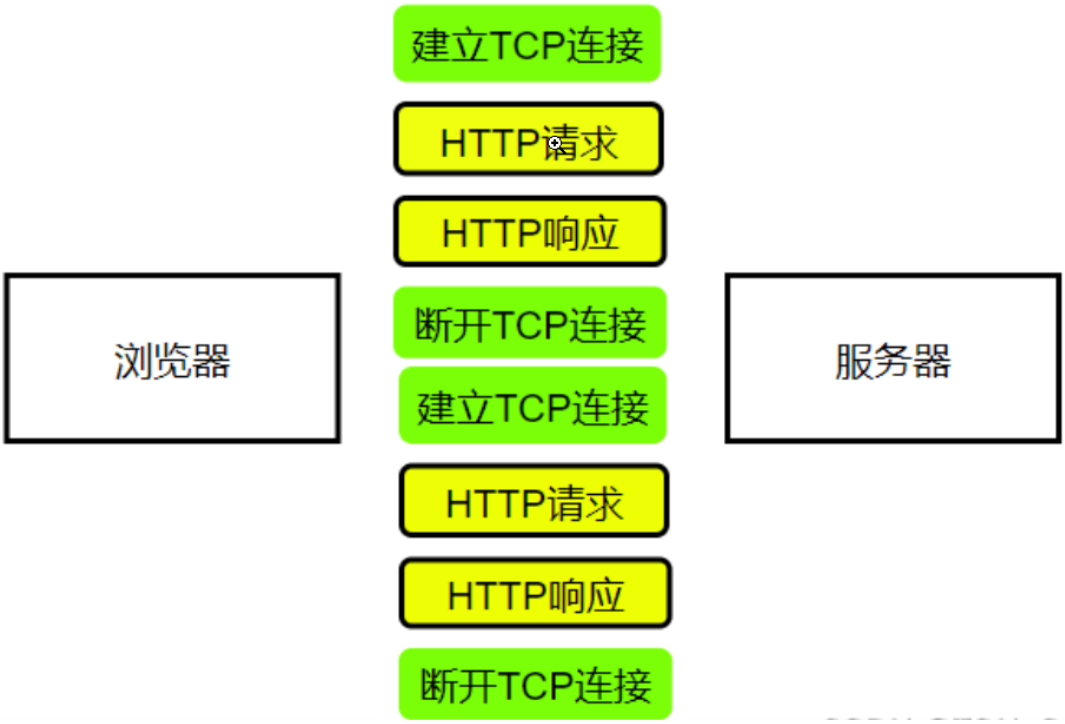
内联，将图片的原始数据嵌入在Css文件里面的URL里减少网络请求次数

二、HTTP1.0

HTTP协议的第二个版本，是第一个在通讯中指定版本号的HTTP协议版本。

特点：

- 每次请求都必须新建一次连接，必须通过TCP的三次握手才能开始传输数据，连接结束之后还要经历四次挥手。
- 不跟踪每个浏览器的历史请求



缺点：

连接无法复用-每次请求都需要建立一个TCP连接，费时费力

队头阻塞，下一个请求必须在前一个请求响应到达后发送。如果某请求一直不到达，那么下一个请求就一直不发送。（高延迟--带来页面加载速度的降低）

每下载文件时都需要建立TCP连接、传输数据和断开连接这样的步骤，无疑会增加大量无谓的开销，因此HTTP1.1增加了持久连接的方法。

三、HTTP1.1

在HTTP1.1中，默认支持长连接，即在一个TCP连接上可以传送多个HTTP请求和响应，减少了建立和关闭连接的消耗和延迟

特点：

支持长连接**，通过Keep-Alive保持HTTP连接不断开，避免重复建立TCP连接

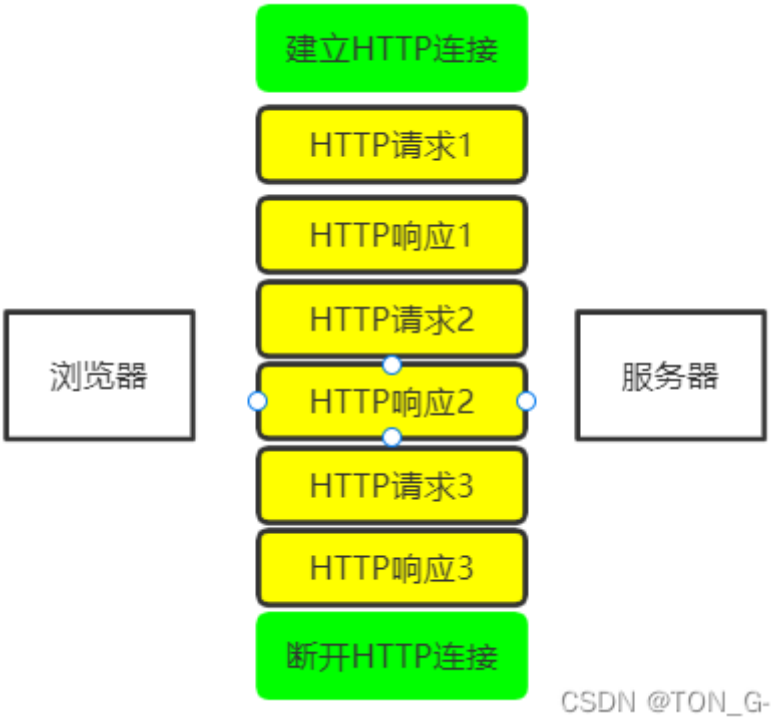
管道化，通过长连接实现在一个连接中传输多个文件

加入缓存处理（新字段cache-control）

支持断点续传

增加了Host字段，实现了在一台WEB服务器上可以同一个IP地址和端口号上使用不同的主机名来创建多个虚拟WEB站点

并且添加了其他请求方法：put、delete、options...

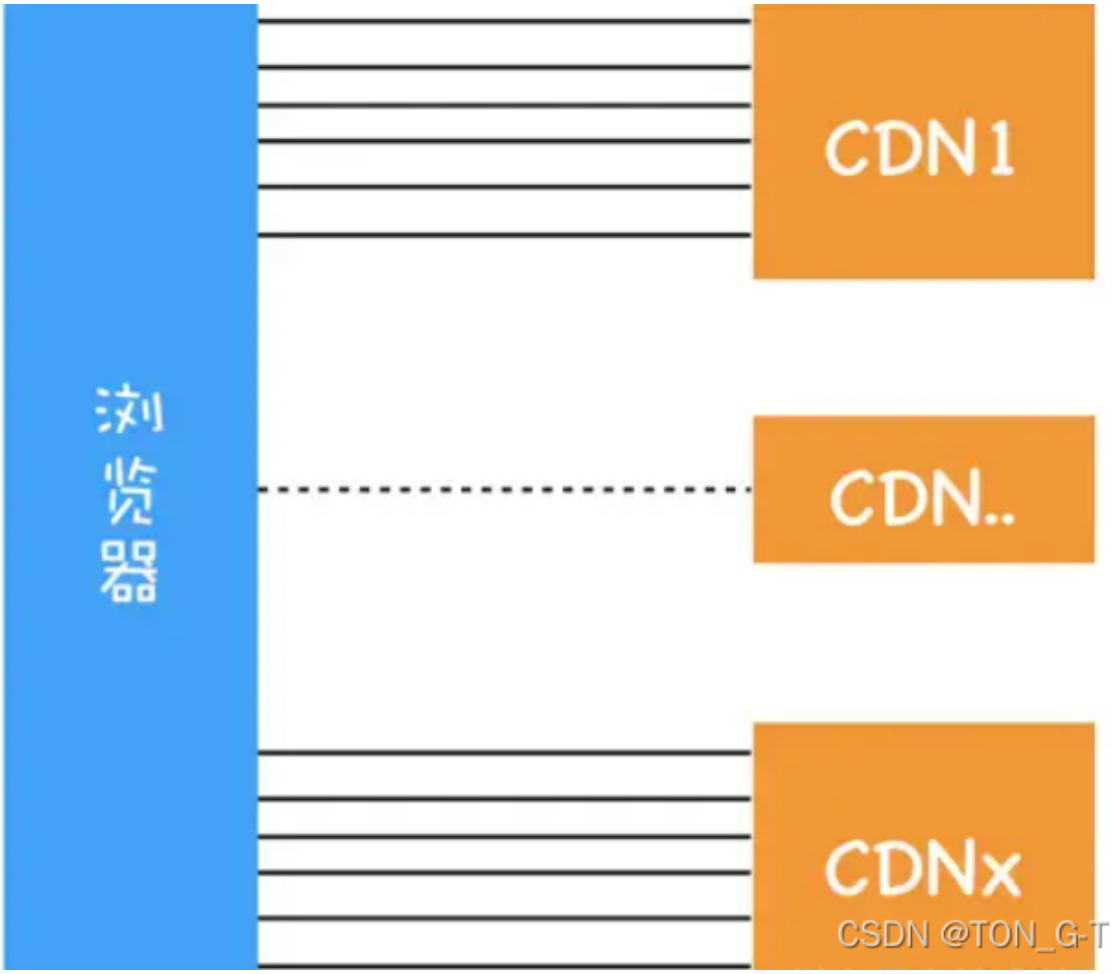


改进和优化：

不成熟的HTTP管线化： HTTP1.1试图用管线化技术来解决队头阻塞问题。HTTP1.1中的管线化是指将多个HTTP请求整批提交给服务器的技术，虽然可以整批发送请求，但是服务器依然需要根据请求顺序来回复浏览器的请求。

对动态生成的内容提高了好的支持：HTTP1.0中，需在响应头中设置完整的数据大小，这样浏览器就可以根据设置的数据大小来接收数据。现在很多页面数据都是动态生成的，HTTP1.1通过引入Chunk transfer机制来解决此问题，服务器将数据分割成若干个任意大小的数据块，每个数据块发送时会附上上个数据块的长度，最后一个0长度的块作为发送数据完成的标志

浏览器为每个域名同时维护6个TCP持久连接，使用CDN实现域名分片机制，为网络做了很大优化，大大提高了页面的下载速度



缺点：

高延迟--队头阻塞

无状态特性--阻碍交互

明文传输--不安全

不支持服务端推送

无状态特性：（带来巨大头部）协议对于连接状态没有记忆能力。纯净的HTTP是没有cookie等机制的，每一个连接都是一个新的连接。上一次请求验证了用户名密码，而下一次请求服务器并不知道它与上一条请求有何关联。

不安全性：传输内容没有加密，中途可能被篡改和劫持

HTTP1.1对带宽的利用率并不理想（带宽是指每秒最大能发送或接收的字节数），HTTP1.1很难将带宽装满：

一是因为TCP的慢启动（为了减少网络拥塞的策略），一个TCP连接建立后，就进入发送数据状态，刚开始TCP协议会采用一个非常慢的速度去发送数据，而页面中常用的一些关键资源文件本来就不大，如HTML文件、CSS文件和JavaScript文件，通常这些文件在建立好连接之后就要发起请求，但这个过程慢启动，耗费时间就要更多。

二是因为同时开启了多条TCP连接，那么这些连接会竞争固定的带宽

三是队头阻塞问题

HTTP2则在此基础上做出了改进

四、HTTP2

HTTP2是基于SPDY，专注于性能，最大的一个目标是在用户和网站键只用一个连接 。

特性：

二进制传输：将请求和响应数据分为更小的帧，并且采用二进制编码

Header压缩：采用HPACK算法压缩头部，同时同一个域名下的两个请求，只会发送差异数据，减少冗余的数据传输，降低开销

多路复用：同一个域名下所有通信都是单个连接上完成，单个连接可以承载任意数量的双向数据流，数据流以消息形式发送，而消息由一个或多个帧组成，可以乱序发送

服务端推送：服务端可以新建“流”主动向客户端发送消息，提前推送客户端需要的静态资源，减少等待延迟

提高安全性：HTTP2也是明文的，只不过格式是二进制的，但HTTP2都是https协议的，跑在TSL上面。

虽然TCP有问题（慢启动），但暂时未有能力换掉，所以想办法规避TCP慢启动和TCP连接之间的竞争问题。HTTP2采用一个域名只使用一个TCP长连接来传输数据，这样整个页面资源的下载过程只需一次慢启动，也避免了多个TCP连接竞争带宽。同时实现资源的并行请求，解决队头阻塞问题。

HTTP2添加了一个二进制分帧层：

HTTP2的请求和连接过程：

首先，浏览器准备好请求数据，包括了请求行、请求头等信息，如果是POST方法还有请求体。

这些数据经过二进制分帧层处理之后，会被转换为一个个带有请求ID编号的帧，通过协议栈将这些帧发送给服务器。

服务器收到所有帧后，会将所有相同ID的帧合并为一条完整的请求信息

服务器处理该条请求，并将处理的请求行、响应头和分别送至二进制分帧层

同样，二进制分帧层会将这些响应数据转换为一个个带有请求ID编号的帧，经协议栈发送到浏览器

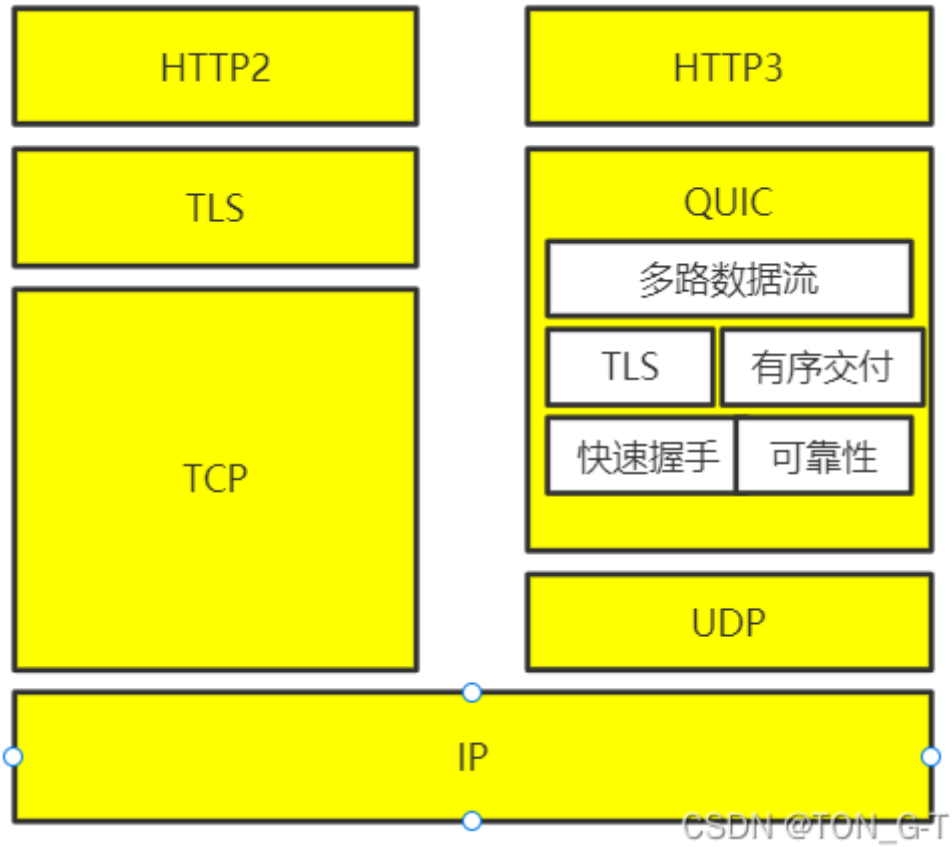
浏览器接受响应帧后，会根据ID编号将帧的数据提交给对应的请求

HTTP2仍存在问题：TCP+TLS建立连接的时间是主要瓶颈：没有从根本上解决队头阻塞问题，一旦遇到丢包，TCP协议还是会重新发送数据。我们知道在HTTP/2中，多个请求是跑在一个TCP管道中的，如果其中任意一路数据流中出现了丢包的情况，那么就会阻塞该TCP连接中的所有请求。这不同于HTTP/1.1，使用HTTP/1.1时，浏览器为每个域名开启了6个TCP连接，如果其中的1个TCP连接发生了队头阻塞，那么其他的5个连接依然可以继续传输数据。

五、HTTP3

HTTP3甩掉TCP、TSL的包袱，构建高效网络QUIC协议。

HTTP3选择了UDP协议，基于UDP实现了类似TCP的多路数据流、传输可靠性等功能，将这套功能称为QUIC协议。



特性：

- 基于UDP协议改造，实现了快速握手
- 集成了TLS的加密功能
- 多路复用，彻底解决了头阻塞问题（一个物理连接上可以有多个独立的逻辑数据流，实现了数据流的单独传输）
- 实现了类似TCP的流量控制、传输可靠性的功能

总结:

HTTP/1：最早的建立的，它建立在TCP协议之上，对同一服务的每个请求都需建立独立的TCP连接(三次握手四次挥手)。

HTTP/1.1：引入了一种持久连接机制（keep alive），同一连接可在多个请求上复用。 持久连接减少了请求延迟，因为客户端不需要对每个请求都发起昂贵的TCP三次握手，添加了HTTP管道，从理论上讲，这使得客户端可以发送多个请求在等待每个响应之前，必须与请求相同的顺序接收响应，正确实现起来很难，并且许多中间代理服务器无法正确处理管道，最终在许多浏览器中删除此支持。

对头阻塞：相同连接上的后续请求，必须等待先前的请求完成。如果由于任何原因列如数据包丢失阻塞了请求，同一连接的所有后续请求都会受到影响。

为了使加载性能能保持在可接受的水平，浏览器通常与同一服务器保持多个TCP连接并发的发送请求。

HTTP/2：引入HTTP流，其中多个请求流，可以通过单个TCP连接发送到同一服务器，不同于HTTP/1.1管道，每个流彼此孤立，而且不需要按顺序发送或接收。HTTP/2解决了应用层的队头阻塞问题，但在TCP传输层问题依然存在。另外，HTTP/2引入了推送功能，允许服务器在可用的新数据时向客户端发送更新，不需要客户端拉取。

HTTP/3：使用了一个名为QUIC的新协议，替换了TCP作为传输层协议。QUIC基于UDP协议，它在传输层引入流作为“一等公民”，QUIC流共享相同的快速连接，因此不需要欸外的握手，来创建新连接。QUIC流是独立交付的，在大多数情况下，丢包仅影响一个流，并不影响其他流，这就是QUIC在传输层消除对头阻塞的方法。

扩展：QUIC是为了移动互联网重度使用而设计的，携带智能手机的人们经常从一个网络切换到另外一个网络。如果使用TCP，将一个连接从一个网络切换到另外一个网络是很慢的。QUIC实现了一个称为连接ID的概念，它允许连接在IP地址和网络接口之间快速可靠的移动。

2，整理apache与nginx之间的区别

一、apache与nginx的区别：

1.二者最核心的区别在于apache是同步多进程模型，一个连接对应一个进程；nginx是异步的，多个连接（万级别）可以对应一个进程 。nginx处理静态文件好,耗费内存少.但无疑apache仍然是目前的主流,有很多丰富的特性.所以还需要搭配着来.当然如果能确定nginx就适合需求,那么使用nginx会是更经济的方式。

2.nginx的负载能力比apache高很多。最新的服务器也改用nginx了。而且nginx改完配置能-t测试一下配置有没 有问题。

3.apache重启的时候发现配置出错了，会很崩溃，改的时候都会非常小心翼翼现在看有好多集群站，前端nginx抗并发，后端apache集群， 配合的也不错。

4.nginx处理动态请求是鸡肋，一般动态请求要apache去做，nginx只适合静态和反向。

5.从经验来看，nginx是很不错的前端服务器，负载性能很好，nginx，用webbench模拟10000个静态文件请求毫不吃力。 apache对php等语言的支持很好，此外apache有强大的支持网络，发展时间相对nginx更久，bug少但是apache有先天不支持多核心处理负载鸡肋的缺点，建议使用nginx做前端，后端用apache。大型网站建议用nginx自代的集群功能。

6.大部分情况下nginx都优于APACHE，比如说静态文件处理、PHP-CGI的支持、反向代理功能、前端Cache、维持连接等等。在Apache+PHP（prefork）模式下，如果PHP处理慢或者前端压力很大的情况下，很容易出现Apache进程数 飙升，从而拒绝服务的现象。

7.Apache在处理动态有优势，Nginx并发性比较好，CPU内存占用低，如果rewrite频繁，那还是Apache吧!

8.一般来说，需要性能的web 服务，用nginx 。如果不需要性能只求稳定，那就apache 吧。

二、apache与nginx优缺点比较

1.nginx相对于apache的优点：
轻量级，同样web 服务，比apache 占用更少的内存及资源；
抗并发，nginx 处理请求是异步非阻塞的，而apache 则是阻塞型的，在高并发下nginx 能保持低资源低消耗高性能；
高度模块化的设计，编写模块相对简单；
社区活跃，各种高性能模块出品迅速啊；
Nginx本身就是一个反向代理服务器，Nginx支持7层负载均衡；Nginx可能会比apache支持更高的并发，
nginx配置文件写的很简洁，正则配置让很多事情变得简单运行效率高，占用资源少，代理功能强大，很适合做前端响应服务器！

2.apache 相对于nginx 的优点：
rewrite，比nginx 的rewrite 强大；
模块超多，基本想到的都可以找到；
少bug，nginx 的bug 相对较多；
超稳定，Aapche依然是大部分公司的首先，因为其成熟的技术和开发社区已经 也是非常不错的性能。

3，nginx一键安装脚本。

```
#!/bin/bash
Yum (){
yum install yum-utils -y
cat > /etc/yum.repos.d/nginx.repo <<'EOF'
[nginx-stable]
name=nginx stable repo
baseurl=http://nginx.org/packages/centos/$releasever/$basearch/
gpgcheck=1
enabled=1
gpgkey=https://nginx.org/keys/nginx_signing.key
module_hotfixes=true

[nginx-mainline]
name=nginx mainline repo
baseurl=http://nginx.org/packages/mainline/centos/$releasever/$basearch/
gpgcheck=1
enabled=0
gpgkey=https://nginx.org/keys/nginx_signing.key
module_hotfixes=true
EOF
yum-config-manager --enable nginx-mainline -y
echo 'y'| yum install nginx -y

}


Source_code (){
yum install -y openssl-devel openssl pcre pcre-devel gcc gcc-c++ zlib zlib-devel
cd /opt
wget https://nginx.org/download/nginx-1.22.0.tar.gz
tar -xvf nginx-1.22.0.tar.gz
cd nginx-1.22.0
useradd -r -g nginx -s /sbin/nologin -d /usr/local/nginx -M nginx
```

```
./configure --prefix=/usr/local/nginx --sbin-path=/usr/sbin/nginx --conf-
path=/etc/nginx/nginx.conf --with-http_ssl_module --with-http_stub_status_module
--with-threads --with-file-aio
make && make install


}

while true
do
echo -e "\e[36m
_______________________________
|                              |
|          Mongodb             |
|      1.yum安装             |
|      2.源码安装              |
|                        |
|                              |
|      `date "+%F|%H:%M:%S"`     |
|   请用source指令启动该脚本   |
|   9.退出程序                 |
|______________________________|
(\__/) ||
(•ㅅ•) ||
/    づv\e[Om"
read -p "请输入你的指示:" I
case $I in
1|yum安装)
    Yum
    continue
    ;;
2|源码安装)
    Source_code;
    continue
    ;;
9|退出程序)
    echo "谢谢使用"
    break
    ;;
    *)
    exit
    esac
done
```

4，整理https协议，然后整理下在浏览器中敲下一条url发生了什么？（过程）

一、HTTPS

HTTP协议中没有加密机制,但可以通 过和 SSL(Secure Socket Layer, **安全套接层**)或 TLS(Transport Layer Security, **安全层传输协议**)的组合使用,加密HTTP的通信内容。属于通信加密，即在整个通信线路中加密。

```
HTTP + 加密 + 认证 + 完整性保护 = HTTPS（HTTP Secure ）
```

HTTPS 采用共享密钥加密（对称）和公开密钥加密（非对称）两者并用的混合加密机制。若密钥能够实现安全交换,那么有可能会考虑仅使用公开密钥加密来通信。但是公开密钥加密与共享密钥加密相比,其处理速度要慢。

所以应充分利用两者各自的优势, 将多种方法组合起来用于通信。 在交换密钥阶段使用公开密钥加密方式,之后的建立通信交换报文阶段 则使用共享密钥加密方式。

1.浏览器将自己支持的一套加密规则发送给网站。

服务器获得浏览器公钥

2.网站从中选出一组加密算法与HASH算法，并将自己的身份信息以证书的形式发回给浏览器。证书里面包含了网站地址，加密公钥，以及证书的颁发机构等信息。

浏览器获得服务器公钥

3.获得网站证书之后浏览器要做以下工作：
(a). 验证证书的合法性（颁发证书的机构是否合法，证书中包含的网站地址是否与正在访问的地址一致等），如果证书受信任，则浏览器栏里面会显示一个小锁头，否则会给出证书不受信的提示。
(b). 如果证书受信任，或者是用户接受了不受信的证书，浏览器会生成一串随机数的密码（接下来通信的密钥），并用证书中提供的公钥加密（共享密钥加密）。
(c) 使用约定好的HASH计算握手消息，并使用生成的随机数对消息进行加密，最后将之前生成的所有信息发送给网站。

浏览器验证 -> 随机密码 服务器的公钥加密 -> 通信的密钥 通信的密钥 -> 服务器

4.网站接收浏览器发来的数据之后要做以下的操作：
(a). 使用自己的私钥将信息解密取出密码，使用密码解密浏览器发来的握手消息，并验证HASH是否与浏览器发来的一致。
(b). 使用密码加密一段握手消息，发送给浏览器。

服务器用自己的私钥解出随机密码 -> 用密码解密握手消息（共享密钥通信）-> 验证HASH与浏览器是否一致（验证浏览器）

HTTPS的不足

5.加密解密过程复杂，导致访问速度慢

6.加密需要认向证机构付费

7.整个页面的请求都要使用HTTPS

二、敲下一条url发生了什么？（过程）

域名解析 --> 发起TCP的3次握手 --> 建立TCP连接后发起http请求 --> 服务器响应http请求，浏览器得到html代码 --> 浏览器解析html代码，并请求html代码中的资源（如js、css、图片等） --> 浏览器对页面进行渲染呈现给用户

再Chrome浏览器上输入www.baidu.com

一.域名解析

首先Chrome浏览器会解析www.baidu.com这个域名对应的IP地址。

1.Chrome浏览器会首先搜索**浏览器的DNS缓存**（缓存时间比较短，TTL默认是1000，且只能容纳1000条缓存），看自身的缓存中是否有www.baidu.com对应的条目，而且没有过期，如果有且没有过期则解析到此结束。

2.如果浏览器自身的缓存里面没有找到对应的条目，那么Chrome会搜索**操作系统的DNS缓存**,如果找到且没有过期则停止搜索解析到此结束。

注：怎么查看操作系统的DNS缓存，以Windows系统为例，可以在命令行下使用 ipconfig /displaydns 来进行查看

3.如果在Windows系统的DNS缓存也没有找到，那么尝试读取hosts文件（位于C:\Windows\System32\drivers\etc），看看这里面有没有该域名对应的IP地址，如果有则解析成功。

4.如果在hosts文件中也没有找到对应的条目，浏览器就会发起一个DNS的系统调用，就会向本地配置的首选DNS服务器（一般是电信运营商提供的，也可以使用像Google提供的DNS服务器）发起域名解析请求（通过的是UDP协议向DNS的53端口发起请求，这个请求是**递归的请求**，也就是运营商的DNS服务器必须得提供给我们该域名的IP地址），运营商的DNS服务器首先查找自身的缓存，找到对应的条目，且没有过期，则解析成功。如果没有找到对应的条目，则有运营商的DNS代我们的浏览器发起**迭代DNS解析请求**，它首先是会找根域的DNS的IP地址（这个DNS服务器都内置13台根域的DNS的IP地址），找打根域的DNS地址，就会向其发起请求（请问www.baidu.com这个域名的IP地址是多少啊？），根域发现这是一个顶级域com域的一个域名，于是就告诉运营商的DNS我不知道这个域名的IP地址，但是我知道com域的IP地址，你去找它去，于是运营商的DNS就得到了com域的IP地址，又向com域的IP地址发起了请求（请问www.baidu.com这个域名的IP地址是多少?）,com域这台服务器告诉运营商的DNS我不知道www.baidu.com这个域名的IP地址，但是我知道www.baidu.com这个域的DNS地址，你去找它去，于是运营商的DNS又向www.baidu.com这个域名的DNS地址（这个一般就是由域名注册商提供的，像万网，新网等）发起请求（请问www.baidu.com这个域名的IP地址是多少？），这个时候cnblogs.com域的DNS服务器一查，果真在我这里，于是就把找到的结果发送给运营商的DNS服务器，这个时候运营商的DNS服务器就拿到了www.baidu.com这个域名对应的IP地址，并返回给Windows系统内核，内核又把结果返回给浏览器，终于浏览器拿到了www.baidu.com对应的IP地址，该进行一步的动作了。

注：一般情况下是不会进行以下步骤的，如果经过以上的4个步骤，还没有解析成功，那么会进行如下步骤：

5.操作系统就会查找NetBIOS name Cache（NetBIOS名称缓存，就存在客户端电脑中的），那这个缓存有什么东西呢？凡是最近一段时间内和我成功通讯的计算机的计算机名和Ip地址，就都会存在这个缓存里面。什么情况下该步能解析成功呢？就是该名称正好是几分钟前和我成功通信过，那么这一步就可以成功解析。

6.如果第5步也没有成功，那会查询WINS 服务器（是NETBIOS名称和IP地址对应的服务器）

7.如果第6步也没有查询成功，那么客户端就要进行广播查找

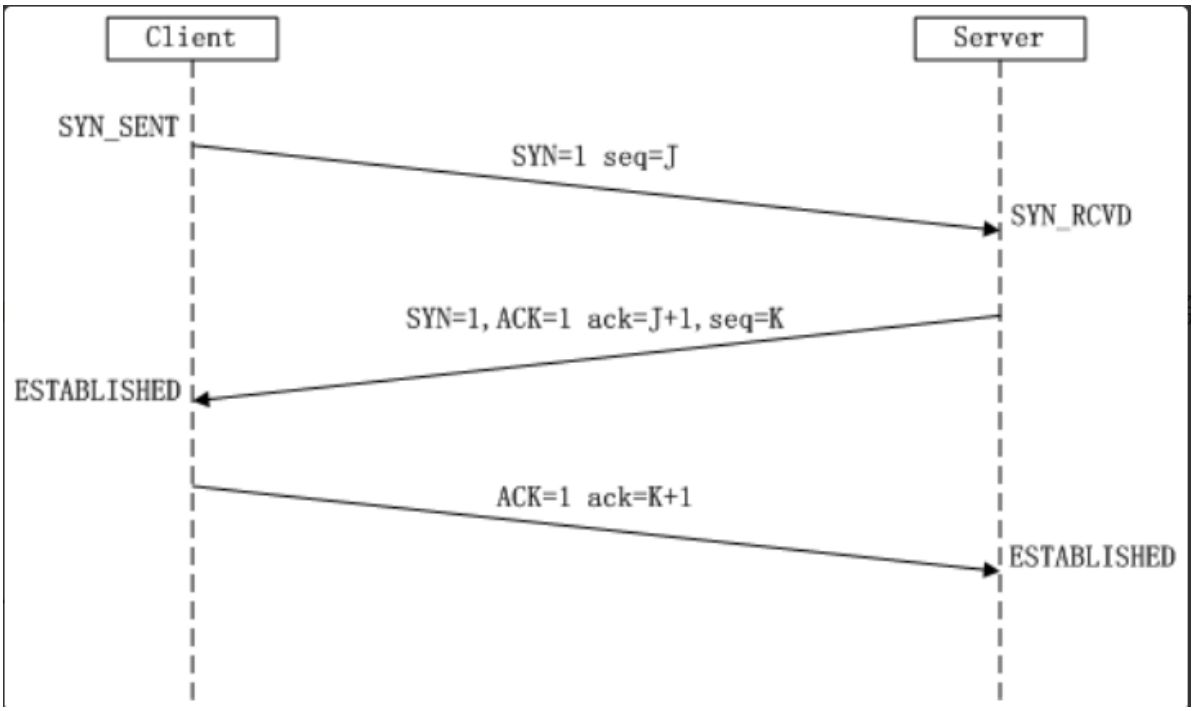
8.如果第7步也没有成功，那么客户端就读取LMHOSTS文件（和HOSTS文件同一个目录下，写法也一样）

如果第八步还没有解析成功，那么这次解析失败，那就无法跟目标计算机进行通信。只要这八步中有一步可以解析成功，那就可以成功和目标计算机进行通信。

二、发起TCP的三次握手

拿到域名对应的IP地址之后，User-Agent（一般是指浏览器）会以一个随机端口（1024 < 端口 < 65535）向服务器的WEB程序（常用的有tomcat,nginx等）80端口发起TCP的连接请求。这个连接请求（原始的http请求经过TCP/IP4层模型的层层封包）到达服务器端后（这中间通过各种路由设备，局域网内除外），进入到网卡，然后是进入到内核的TCP/IP协议栈（用于识别该连接请求，解封包，一层一层的剥开），还有可能要经过Netfilter防火墙（属于内核的模块）的过滤，最终到达WEB程序，最终建立了TCP/IP的连接。如下图：

所谓三次握手（Three-Way Handshake）即建立TCP连接，就是指建立一个TCP连接时，需要客户端和服务端总共发送3个包以确认连接的建立。整个流程如下图所示：

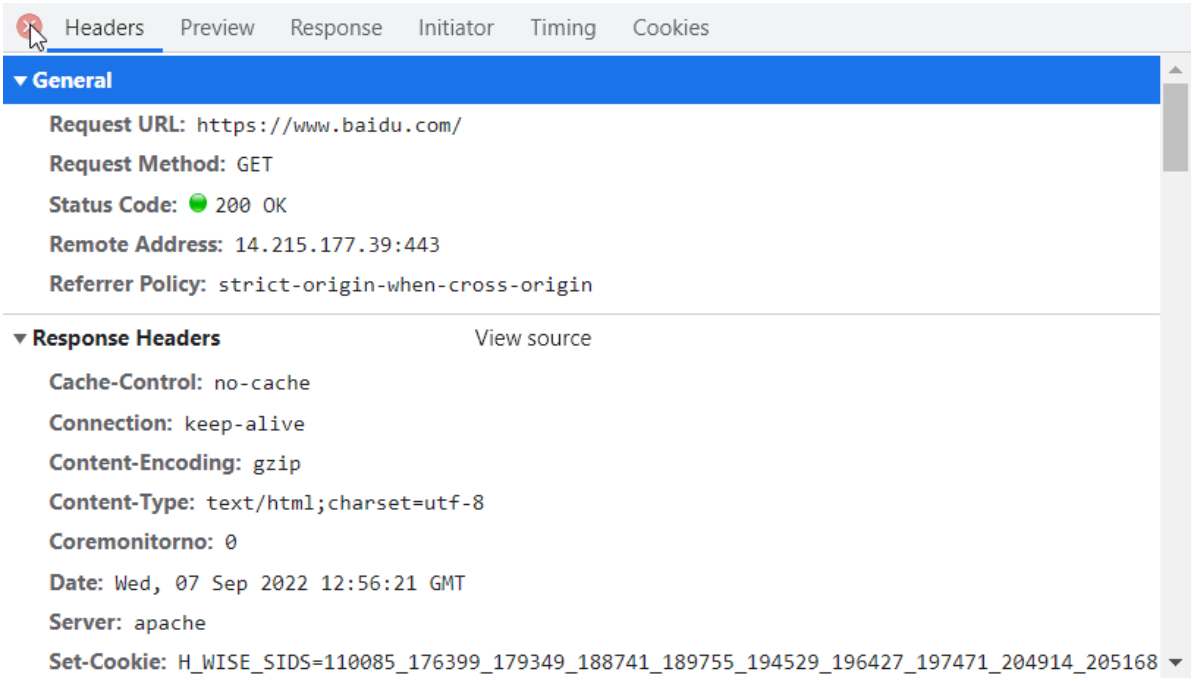


1. 第一次握手：Client将标志位SYN置为1，随机产生一个值seq=J，并将该数据包发送给Server，Client进入SYN_SENT状态，等待Server确认。
2. 第二次握手：Server收到数据包后由标志位SYN=1知道Client请求建立连接，Server将标志位SYN和ACK都置为1，ack=J+1，随机产生一个值seq=K，并将该数据包发送给Client以确认连接请求，Server进入SYN_RCVD状态。
3. 第三次握手：Client收到确认后，检查ack是否为J+1，ACK是否为1，如果正确则将标志位ACK置为1，ack=K+1，并将该数据包发送给Server，Server检查ack是否为K+1，ACK是否为1，如果正确则连接建立成功，Client和Server进入ESTABLISHED状态，完成三次握手，随后Client与Server之间可以开始传输数据了。

三、建立TCP连接后发起http请求

HTTP请求报文的方法是get方式，如果浏览器存储了该域名下的Cookies，那么会把Cookies放入HTTP请求头里发给服务器。

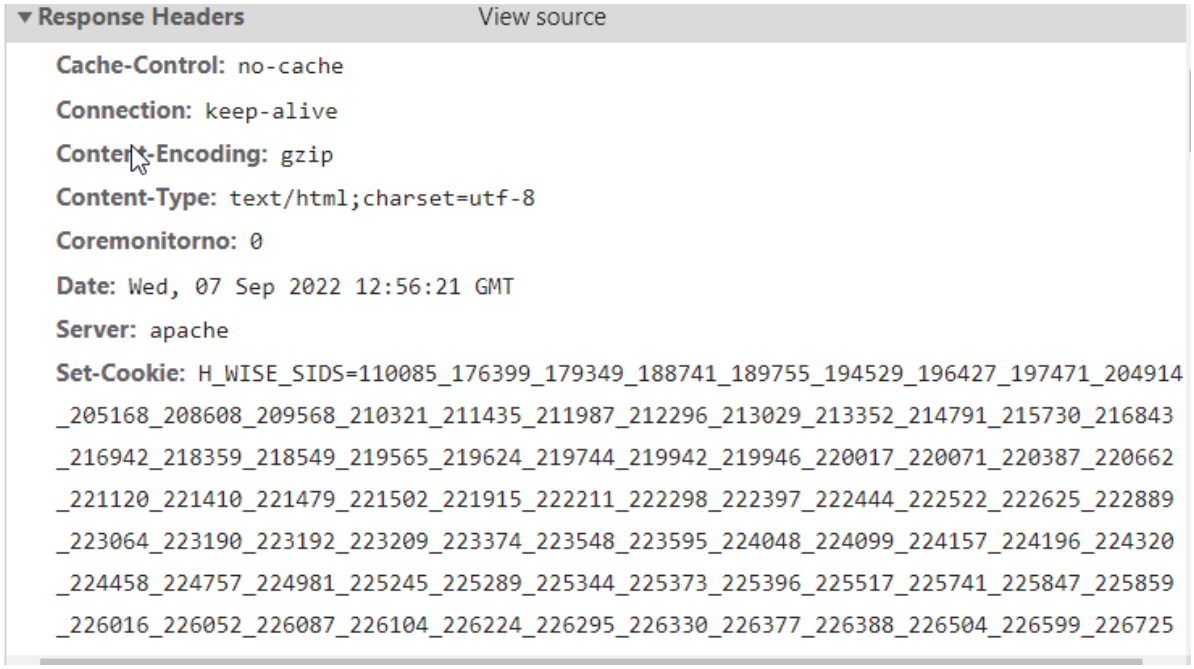
下面是Chrome发起的http请求报文头部信息:



四、服务器端响应http请求，浏览器得到html代码

服务器端WEB程序接收到http请求以后，就开始处理该请求，处理之后就返回给浏览器html文件。

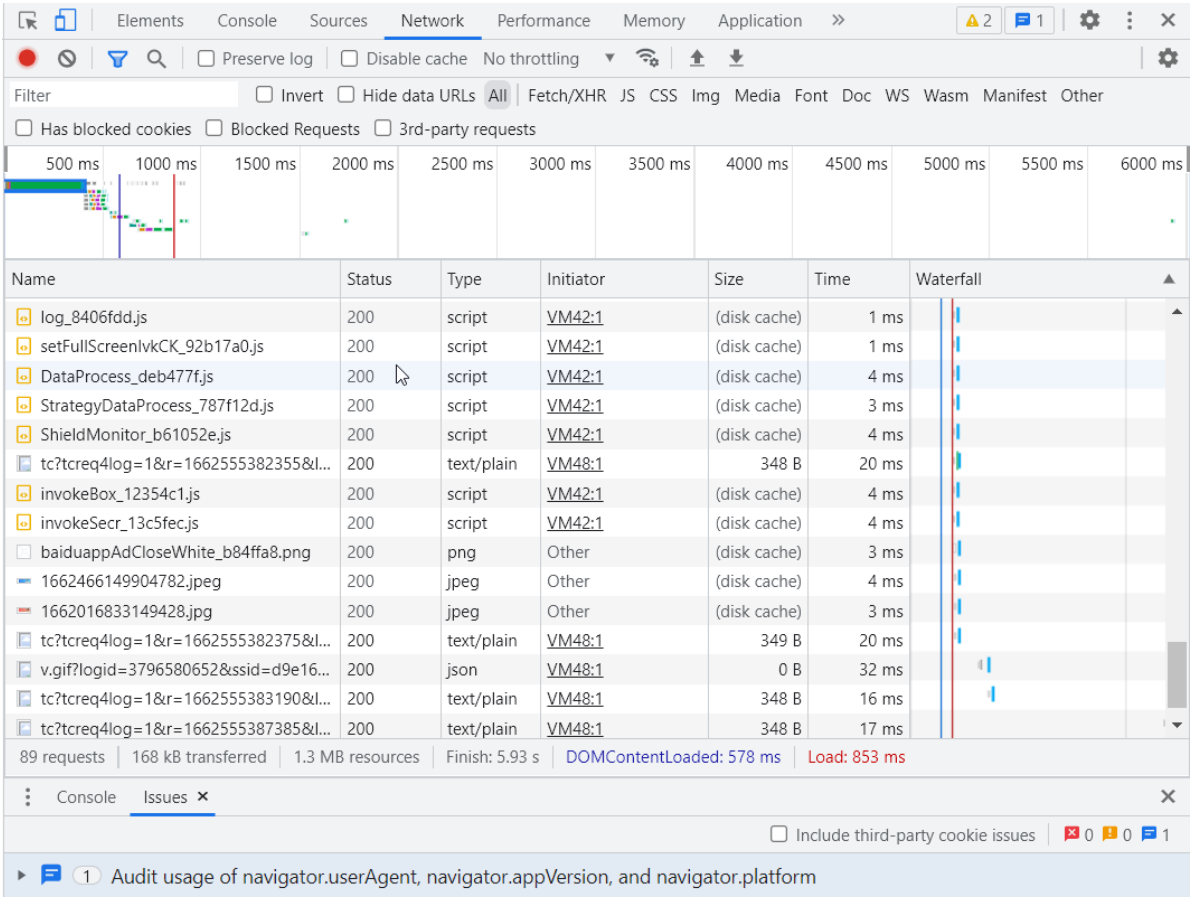
用Chrome浏览器看到的响应头信息：



五、浏览器解析html代码，并请求html代码中的资源

浏览器拿到index.html文件后，就开始解析其中的html代码，遇到js/css/image等静态资源时，就向服务器端去请求下载（会使用多线程下载，每个浏览器的线程数不一样），这个时候就用上keep-alive特性了，建立一次HTTP连接，可以请求多个资源，下载资源的顺序就是按照代码里的顺序，但是由于每个资源大小不一样，而浏览器又多线程请求请求资源，所以从下图看出，这里显示的顺序并不一定是代码里面的顺序。

浏览器在请求静态资源时（在未过期的情况下），向服务器端发起一个http请求（询问自从上一次修改时间到现在有没有对资源进行修改），如果服务器端返回304状态码（告诉浏览器服务器端没有修改），那么浏览器会直接读取本地的该资源的缓存文件。



六、浏览器对页面进行渲染呈现给用户

最后，Chrome浏览器利用自己内部的工作机制，把请求到的静态资源和html代码进行渲染，渲染之后呈现给用户。