一.`OSI`七层模型 (Open System Interconnection)

> `OSI`七层模型，是理想化的模型, 将复杂的流程分解为几个功能实现复杂问题简单化

- 7.应用层 (用户最终的接口) 微信、QQ HTTP\DNS\FTP\TFTP\SMTP\DHCP
- 6.表示层 (数据的表示、安全、压缩)
- 5.会话层 (建立和管理会话)
- 4.传输层 **TCP协议、UDP协议**
- 3.网络层 (路由器) **IP协议 ipv4、ipv6**
- 2.数据链路层 (交换机、网卡)
- 1.物理层 (物理设备，网线、光纤)

> 七层模型是倒着看的，下层是为了上层提供服务的

1.1 IP地址和Mac地址 (ip -> mac 地址只支持局域网的)

- 最终通信都是通过MAC地址通信，网卡中都有自己的mac地址（原则上唯一）
- 通过DHCP协议分配给每台电脑IP，通过ARP协议将IP地址转化成mac地址。

1.2 路由器和交换机

- 交换机：维护MAC地址表 （交换机端口对应的mac地址）不关心ip地址。核心就是交换数据，交换效率高
- 路由器:（lan口、wan口），在不连接wan口的情况下，路由器可以看成是交换机

1.3 网关

- TCP/IP中规定两个子网不能直接通信 （通过子网掩码来区分两个设备是否是同一个子网），我们从内网访问到外网属于两个不同的子网。路由器就充当了网关的角色。so~~~*网关*(Gateway)又称网间连接器、协议转换器 。

> 计算机发送请求192.168.1.16 -> 110.242.68.4（路由器会将源IP进行SNAT + 需要增加端口区分不同主机)

1.4 举例：我们想发快递

- 1.需要建立从发货地址到收获地址的通道（公路）
- 2.找到运输快递的交通工具 （运输快递）
- 3.发快递时需要标记派送地址，传输过程中可能会丢失 （快递会进行再次打包）
- 4.开始传输，我想邮寄一个汽车 （拆包发送，标记序号等）,如果中途丢了。还需要重新发货
- 5.和对方建立联系，准备需要发送的数据

> 总结： 应用层（封装） 传输层（封装） 数据链路层（加头尾封装） 最终以比特流的形式进行传输。对方收到后解封装还原原始数据

二.TCP/IP参考模型 （五层模型）

Transmission Control Protocol/Internet Protocol，传输控制协议/网际协议 ，不仅仅指两个协议(协议簇)

协议就是通信的规则 以`http`协议当做范例来说（协议就是对数据的封装 + 传输）

- 数据链路层、物理层 ： 物理设备
- 网络层:
  - `IP`协议：寻址通过路由器查找，将消息发送给对方路由器，通过`ARP`协议,发送自己的mac地址
  - `ARP` 协议：Address Resolution Protocol 从`ip`地址获取mac地址 （局域网）`RARP` （通信由mac地址通信，通过自己的mac地址，对方的`ip`，获取对方的mac地址）
- 传输层: `TCP`、 `UDP`
- 应用层:`HTTP`、 `DNS`、 `FTP`、 `TFTP`、 `SMTP` 、 `DHCP`

三.DNS协议

`DNS`是Domain Name System的缩写， `DNS`服务器进行域名和与之对应的`IP`地址转换的服务器

- 顶级域名 `.com`、
- 二级域名 `.com.cn`、 三级域名 `www.zf.com.cn`, 有多少个点就是几级域名

> 访问过程：我们访问`zf.com.cn`，会先通过`DNS`**服务器**查找离自己最近的根服务器，通过根服务器找到`.cn`服务器，将`ip`返回给`DNS`服务器， `DNS`服务器会继续像此`ip`发送请求，去查找对应`.cn`下`.com`对应的`ip`，获取最终的`ip`地址。缓存到`DNS`服务器上

四.IP协议 （寻址）

目前主流的是IPV4、IPV6　192.168.1.16

- ipv4地址最大是255.255.255.255, 总共可以编43亿个地址，不够用。所以出现IPV6
- 通过IP地址逐级查找，直到定位到最终的设备, 通过掩码 一层来来进行查找
- 255.0.0.0 255.255.0.0

五.TCP协议 （传输）

`tcp` 传输控制协议 `Transimision Control Protocal` 可靠、面向连接的协议,传输效率低 (在不可靠的`IP`层上建立可靠的传输层)。 TCP提供全双工服务，即数据可在同一时间双向传播。数据是无序地在网络间传递，接收方需要有一种算法在接受到数据后恢复原有的顺序 (一个tcp段就有20个字节的头，如果每次传递一个数据，tcp传输的过程中会粘包)

- 源端口号、目的端口号，指代的是发送方随机端口，目标端对应的端口
- 32位序列号是用于对数据包进行标记，方便重组
- 4位首部长度：单位是字节，4位最大能表示15，所以头部长度最大为60 (一般为20个字节)
- `URG`:紧急新号、`ACK`:确认信号、`PSH`:应该从TCP缓冲区读走数据、`RST`：断开重新连接、`SYN`:建立连接、`FIN`:表示要断开
- 校验和：用来做差错控制，看传输的报文段是否损坏
- 紧急指针：用来发送紧急数据使用

> TCP是提供可靠的网络传输，需要建立连接，流速控制。UDP协议只发送包，无连接协议，速度快。

六.`wireshark`抓包

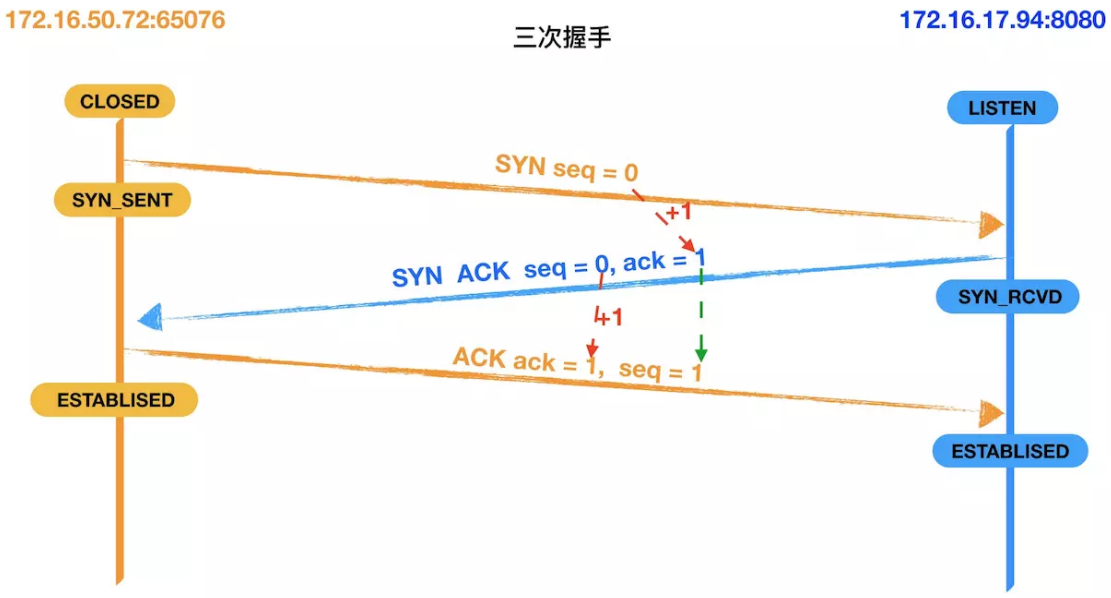
client.js

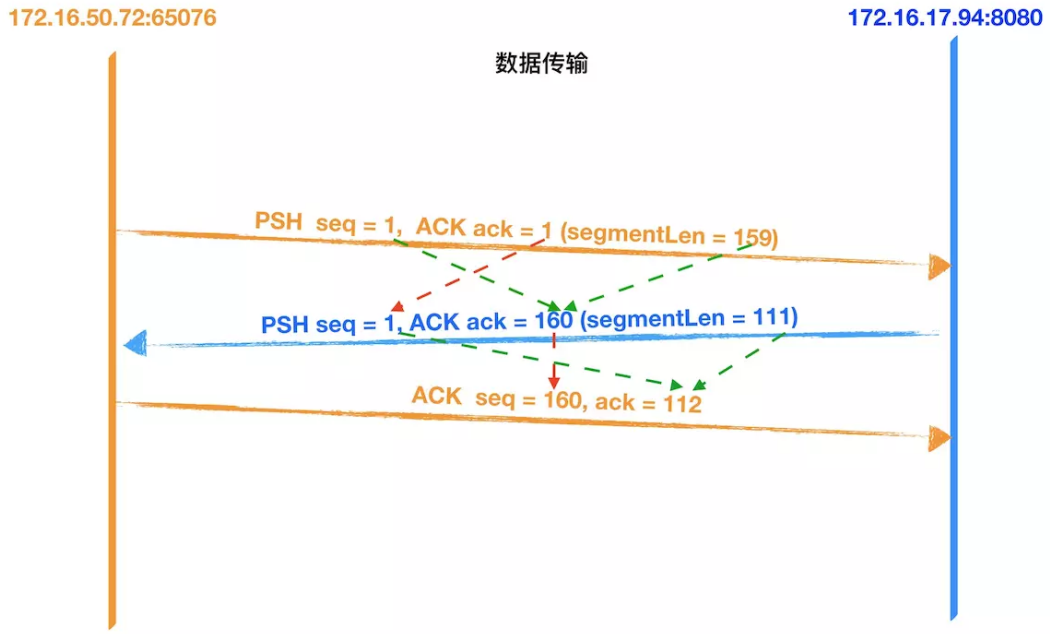
```
const net = require('net');
const socket = new net.Socket();
socket.connect(8080, 'localhost');
socket.on('connect', function(data) {
    socket.write('connect server'); // 可写流
});
socket.on('data', function(data) { // 可读流
    console.log(data.toString())
})
socket.on('error', function(error) {
    console.log(error);
});
```

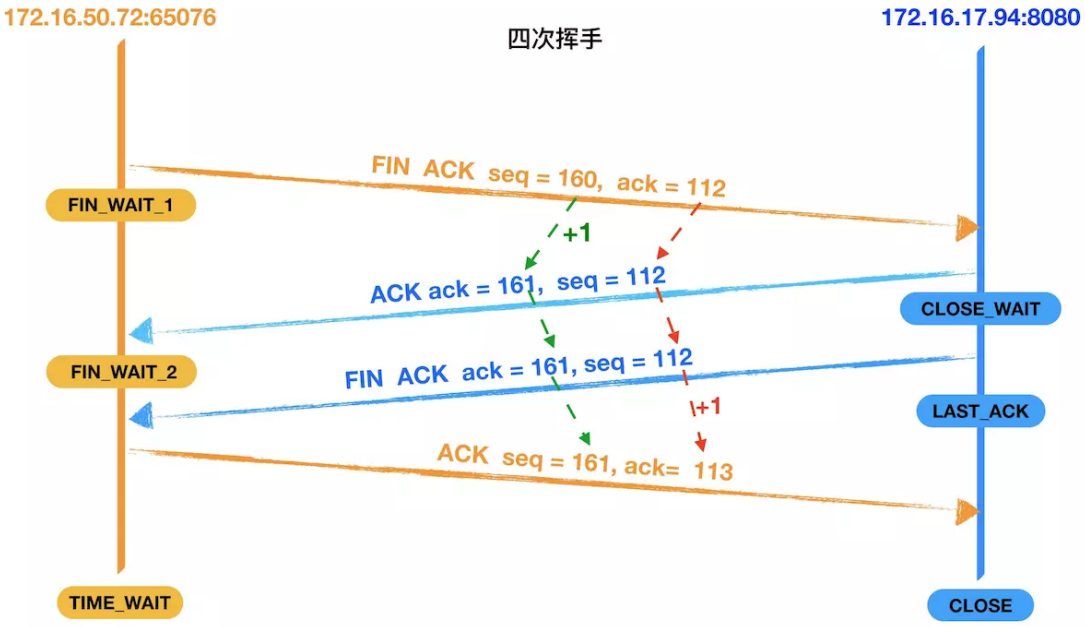
server.js

```
const net = require('net');
const server = net.createServer(function(socket){
    socket.on('data',function (data) { // 可读流
        socket.write('server:hello'); // 可写流
    });
    socket.on('end',function () {
        console.log('客户端关闭')
    })
})
server.on('error',function(err){
    console.log(err);
})
server.listen(8080);
```

6.1.为什么是三次握手四次挥手







- 三次握手
  - 1）我能主动给你打电话吗？ 2）当然可以啊！那我也能给你打电话吗？
  - 3）可以的呢，建立连接成功！
- 四次挥手
  - 1）我们分手吧 2）回复收到分手的信息
  - 3）好吧，分就分吧 4）行，那就到这里了

> 总结：

- TCP是双工的所以，握手需要3次。保证双方达成一致 （建立连接浪费性能）
- 当断开链接时，发送（FIN）时另一方需要马上回复 (ACK), 但此时可能不能立即关闭（有未发送完的数据，还有一些准备断开的操作），所以等待确定可以关闭时在发送 (FIN)

6.2.滑动窗口 （算法题 用滑动窗口）

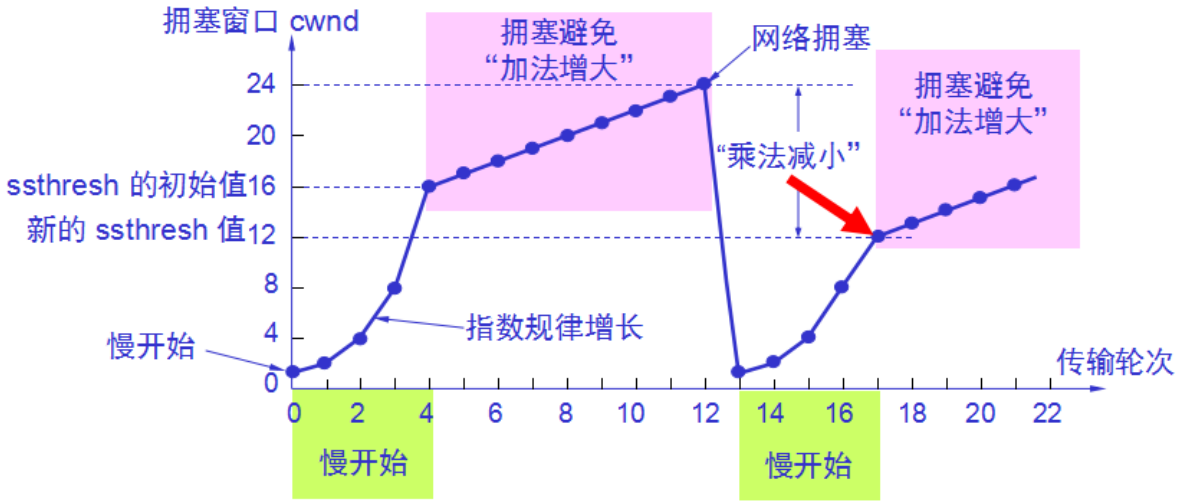
- 滑动窗口：TCP是全双工的，所以发送端有发送缓存区；接收端有接收缓存区，要发送的数据都放 到发送者的缓存区，发送窗口（要被发送的数据）就是要发送缓存中的哪一部分
- 核心是流量控制：在建立连接时，接收端会告诉发送端自己的窗口大小（`rwnd`），每次接收端收到数据后都会再次确认（`rwnd`）大小，如果值为0，停止发送数据。（并发送窗口探测包，持续监测接受端的窗口大小，探测包）
- `Nagle`算法的基本定义是**任意时刻，最多只能有一个未被确认的小段** (TCP内部控制)
- `Cork`算法 当达到`MSS`(Maximum Segment Size)值时统一进行发送 一般用这个值（此值就是帧的大小 - `ip`头 - `tcp`头 = 1460个字节） 但是帧大小在不同网络下是可变的 （如果tcp段很大，那也会在ip层去拆 -》 帧来发）

6.3.慢启动、拥塞避免、快重传和快恢复

> 举例：假设接收方窗口大小是无限的，接收到数据后就能发送`ACK`包，那么传输数据主要是依赖于网络带宽，带宽的大小是有限的。

- TCP 维护一个拥塞窗口`cwnd` （congestion window）变量，在传输过程正没有拥塞就将此值增大。如果出现拥塞（超时重传`RTO(Retransmission TimeOut)`）就将窗口值减少。
- `cwnd < ssthresh` 使用慢开始算法
- `cwnd > ssthresh`使用拥塞避免算法
- ROT时更新`ssthresh`值为当前窗口的一半，更新`cwnd` = 1

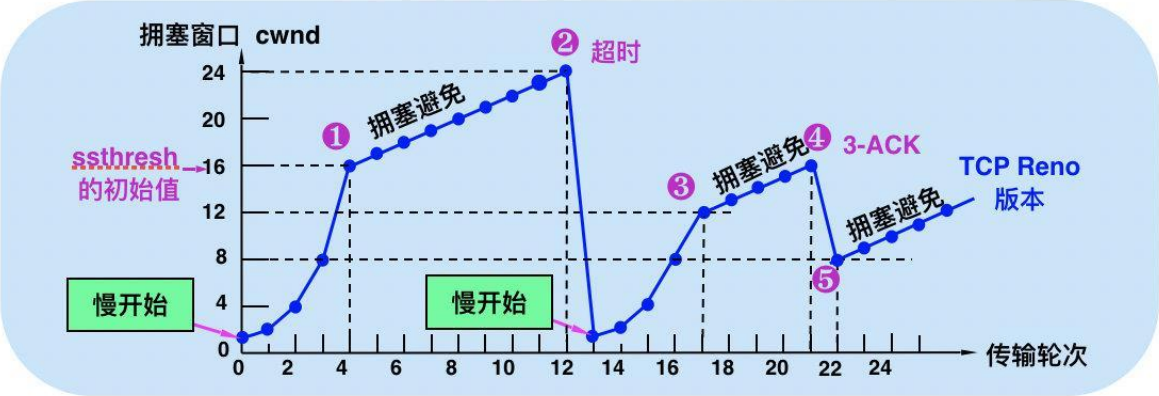
> `Tahoe`版本



- 传输轮次：`RTT` (Round-trip time) ,从发送到确认信号的时间
- `cwnd`控制发送窗口的大小。

> `Reno`算法

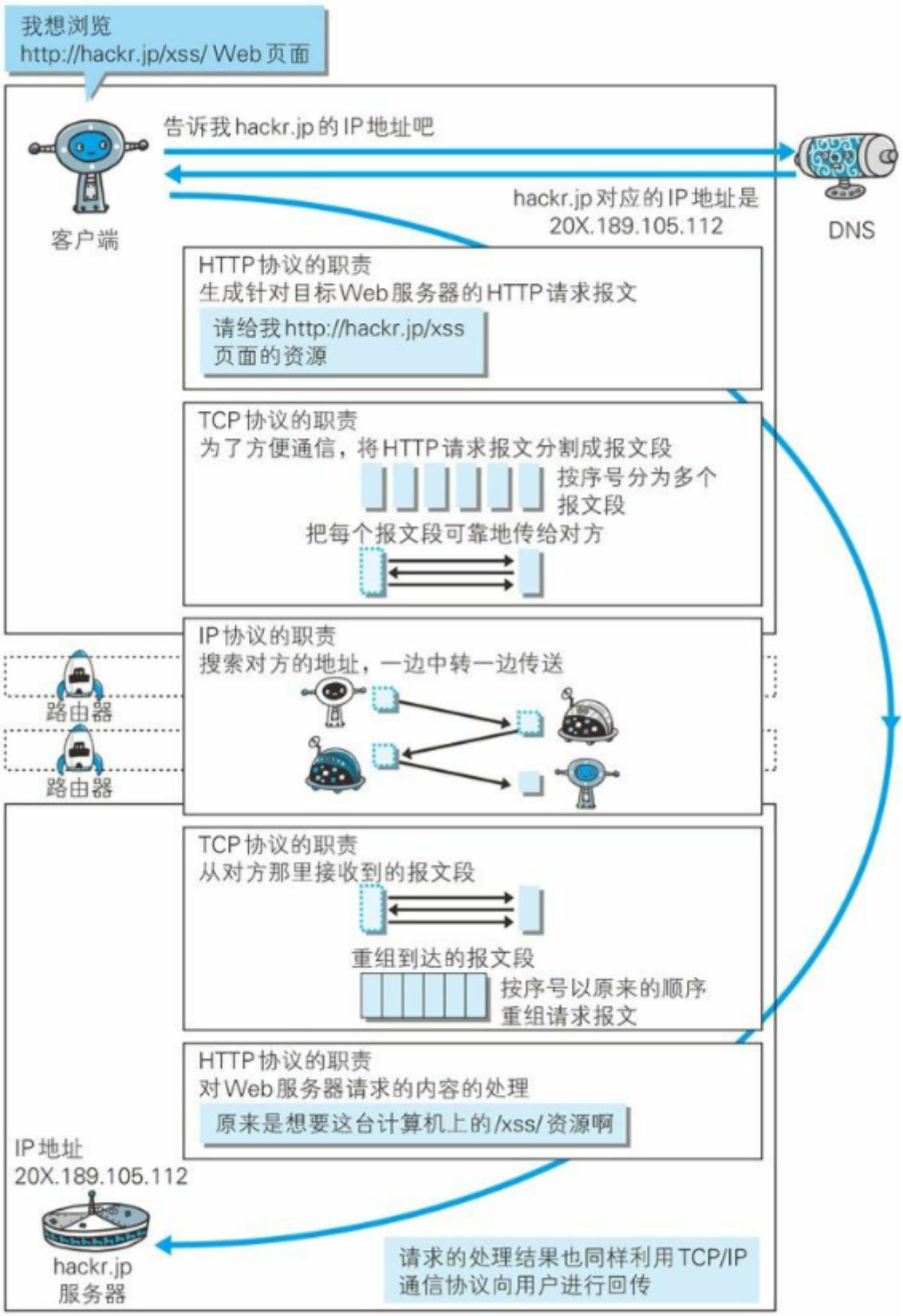
- 快重传，可能在发送的过程中出现丢包情况。此时不要立即回退到慢开始阶段，而是对已经收到的报文重复确认，如果确认次数达到3次，则立即进行重传 **快恢复算法**(减少超时重传机制的出现)，降低重置`cwnd`的频率。
- 更新`ssthresh`值和`cwnd`值为相同



七.HTTP

1.发展历程

- HTTP/0.9 在传输过程中没有请求头和请求体，服务器响应没有返回头信息，内容采用ASCII字符流来进行传输 HTML
- HTTP/1.0 增加了请求头和响应头，实现多类型数据传输
- HTTP/1.1 默认开启持久链接，在一个TCP链接上可以传输多个HTTP请求， 采用管线化的方式（每个域名最多维护6个TCP持久链接）解决**队头阻塞**问题 （服务端需要按顺序依次处理请求）。完美支持数据分块传输（`chunk transfer`），并引入客户端cookie机制、安全机制等。
- HTTP/2.0 解决网络带宽使用率低 （TCP慢启动，多个TCP竞争带宽，队头阻塞）采用多路复用机制（一个域名使用一个TCP长链接，通过二进制分帧层来实现）。头部压缩（`HPACK`）、及服务端推送
- HTTP/3.0 解决TCP队头阻塞问题， 采用`QUIC`协议。`QUIC`协议是基于`UDP`的 （目前：支持和部署是最大的问题）
- HTTP明文传输,在传输过程中会经历路由器、运营商等环节，数据有可能被窃取或篡改 （**安全问题**）



- `http`是不保存状态的协议，使用cookie来管理状态 (登录 先给你cookie 我可以看一下你有没有cookie)
- 为了防止每次请求都会造成无谓的`tcp`链接建立和断开，所以采用保持链接的方式 keep-alive
- 以前发送请求后需要等待并收到响应，才能发下一个，现在都是管线化的方式