OSI七层模型

OSI(Open System Interconnection)是理想化的模型，将网络进行分层（也就是将网络通信的过程分为七个层级），其目的是将复杂的流程简单化，从而实现**分而治之**。

一.地址

> 通信是通过ip地址查找对应的mac来进行通信的。 IP地址是可变的（类似我们收件地址）MAC地址是不可变的。IP地址是逻辑地址而MAC地址是物理地址

1.IP地址

IPV4 网际协议版本4， 地址由 32 位二进制数值组成 例如：192.168.1.1 ， 大概42亿个

`IPV6` 网际协议版本6，地址由 8个16进制数组成，共128位。例如：`2408:8207:788b:2370:9530:b5e7:9c53:ff87` 大约(2^128 = `3.4*10^38`)

2.MAC地址

设备通信都是由内部的网卡设备来进行的，每个网卡都有自己的mac地址（原则上唯一）

二.物理设备

1.物理层

- 中继器：双绞线最大传输距离`100M`，中继器可以延长网络传输的距离，对衰减的信号有放大在生的功能。
- 集线器：多口的中继器，目的是将网络上的所有设备连接在一起，不会过滤数据，也不知道将收到的数据发给谁。（采用的方式就是广播给每个人）

> 可以实现局域网的通信，但是会有安全问题，还会造成不必要的流量浪费。 傻，你就不能记住来过的人嘛？每次都发送？

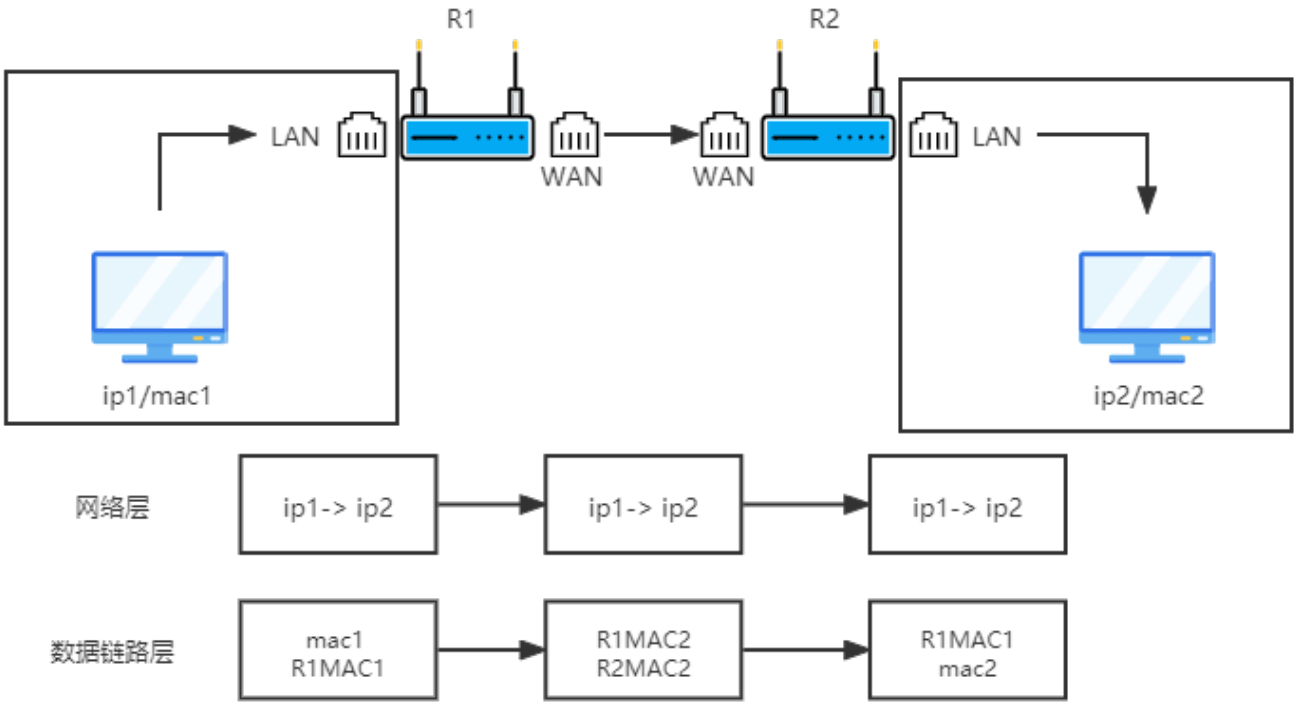
2.数据链路层

- 交换机：交换机可以识别已经连接设备的物理地址（MAC地址）。可以将数据传递到相应的端口上

3.网络层

- 路由器：检测数据的`ip`地址是否属于自己网络，如果不是会发送到另一个网络。没有`wan`口的路由器可以看成交换机。 路由器一般充当网关，路由器会将本地`IP`地址进行NAT(Network Address Translation) 转换为公网IP地址。

> 网关：两个子网之间不可以直接通信，需要通过网关进行转发



三.TCP/IP参考模型

Transmission Control Protocol/Internet Protocol，传输控制协议/网际协议。`TCP/IP` 协议实际上是一系列网络通信协议的统称，最核心的两个协议是`TCP`和`IP`

1.什么是协议?

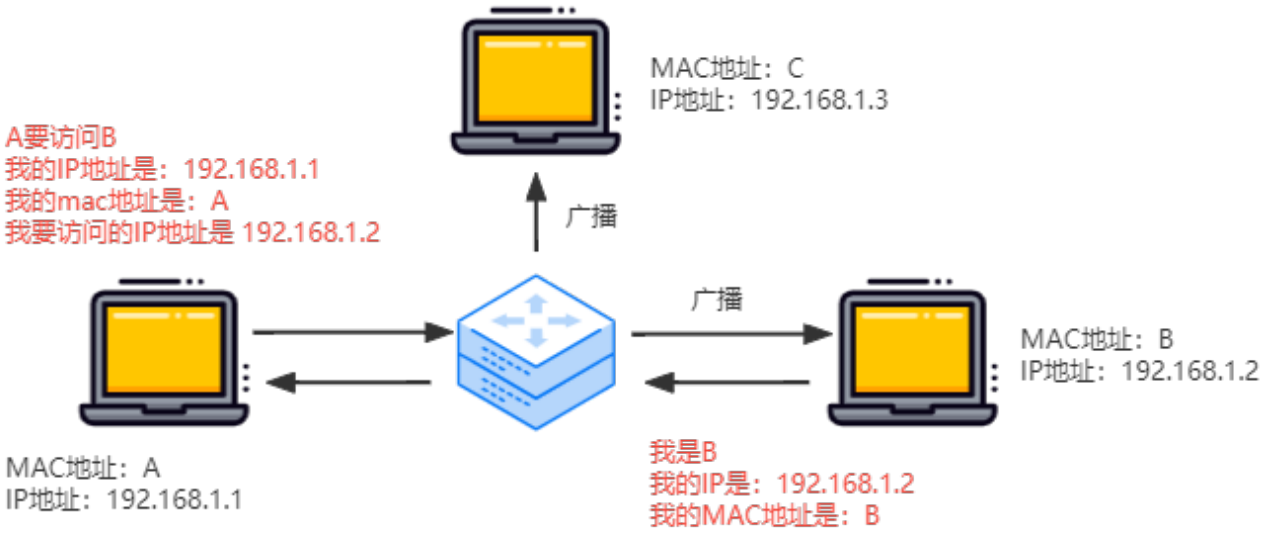
> 协议就是约定和规范。

数据链路层、物理层： 物理设备 (在五层模型中能称之为协议的都在三层以上)

- 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）
- 传输层
  - `TCP`、`UDP`
- 应用层:
  - `HTTP`、`DNS`、`FTP`、`TFTP`、`SMTP` 、`DHCP`

2.ARP协议

> 根据目的IP地址，解析目的mac地址



ARP缓存表		交换机MAC地址表	
Internet 地址	物理地址	端口号	物理地址
192.168.1.2	B	1	A
		2	B
		3	C

> 有了源mac地址和目标mac地址，就可以传输数据包了。

3.DHCP协议

通过DHCP自动获取网络配置信息 （动态主机配置协议 Dynamic Host Configuration

Protocol）我们无需自己手动配置IP

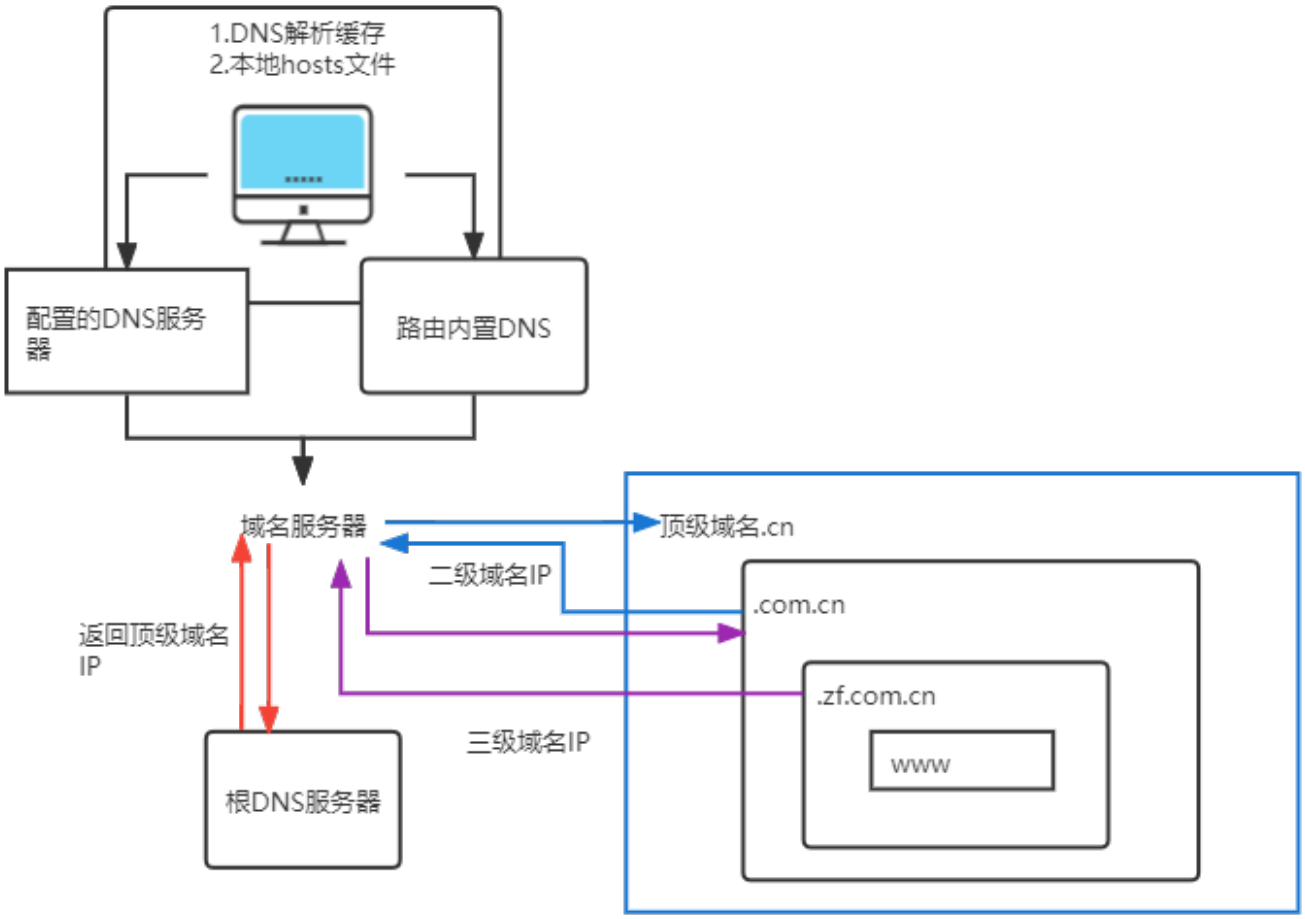
4.DNS协议

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.xxx.com.cn, 有多少个点就是几级域名

访问过程：我们访问xxx.com.cn

- 操作系统里会对 DNS 解析结果做缓存，如果缓存中有直接返回IP地址
- 查找C:\WINDOWS\system32\drivers\etc\hosts 如果有直接返回IP地址
- 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



> DNS服务器会对ip及域名进行缓存

四.TCP和UDP

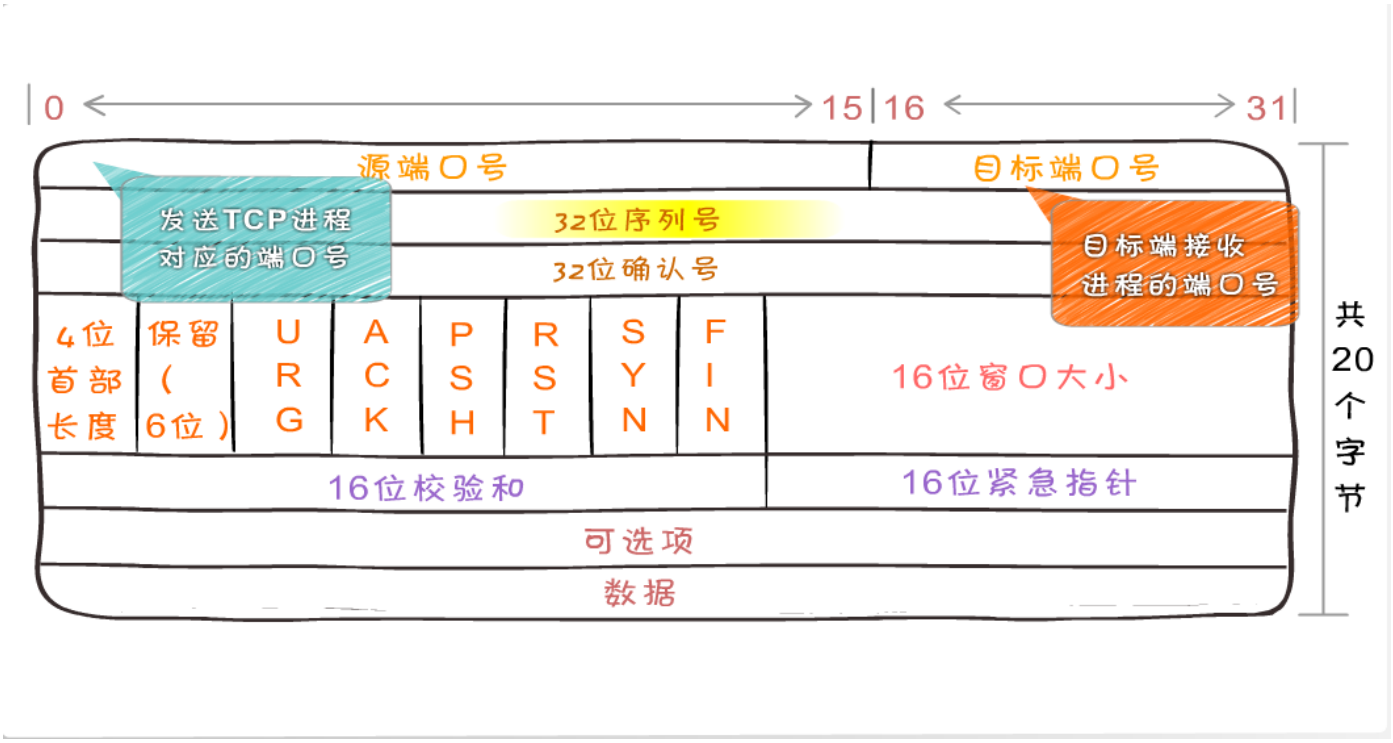
> 两个协议都在传输层，我们经常说TCP 是面向连接的而UDP是面向无连接的。

- UDP发出请求后，不考虑对方是否能接收到、内容是否完整、顺序是否正确。 收到数据后也不会进行通知。
- 首部结构简单，在数据传输时能实现最小的开销

1.TCP

`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。

1）TCP数据格式 （数据帧的 一帧1500 -20ip头部 - 20tcp头部 = 1460数据大小） 理论值超过1460的数据就要去分段传输



- 源端口号、目标端口号，指代的是发送方随机端口，目标端对应的端口 4
- 序列号：32位序列号是用于对数据包进行标记，方便重组 4
- 确认序列号：期望发送方下一个发送的数据的编号 4
- 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 4

- `URG`:紧急信号、`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
- 窗口大小： 当网络通畅时将这个窗口值变大加快传输速度，当网络不稳定时减少这个值。在TCP中起到流量控制作用。
- 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 对数据进行分段打包传输，对每个数据包编号控制顺序。

2.TCP抓包

client.js

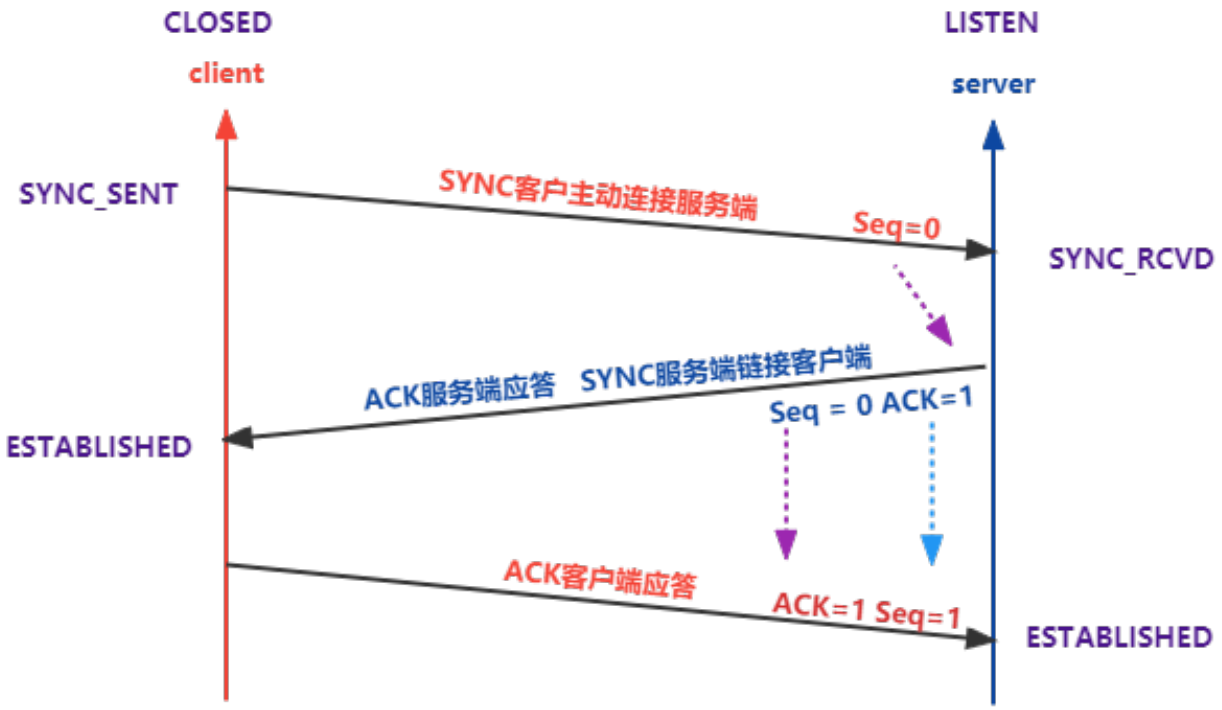
```
const net = require('net');
const socket = new net.Socket();
// 连接8080端口
socket.connect(8080, 'localhost');
// 连接成功后给服务端发送消息
socket.on('connect', function(data) {
    socket.write('hello'); // 浏览器和客户端说
hello
    socket.end()
});
socket.on('data', function(data) {
    console.log(data.toString())
```

```
})
socket.on('error', function(error) {
    console.log(error);
});
```

server.js

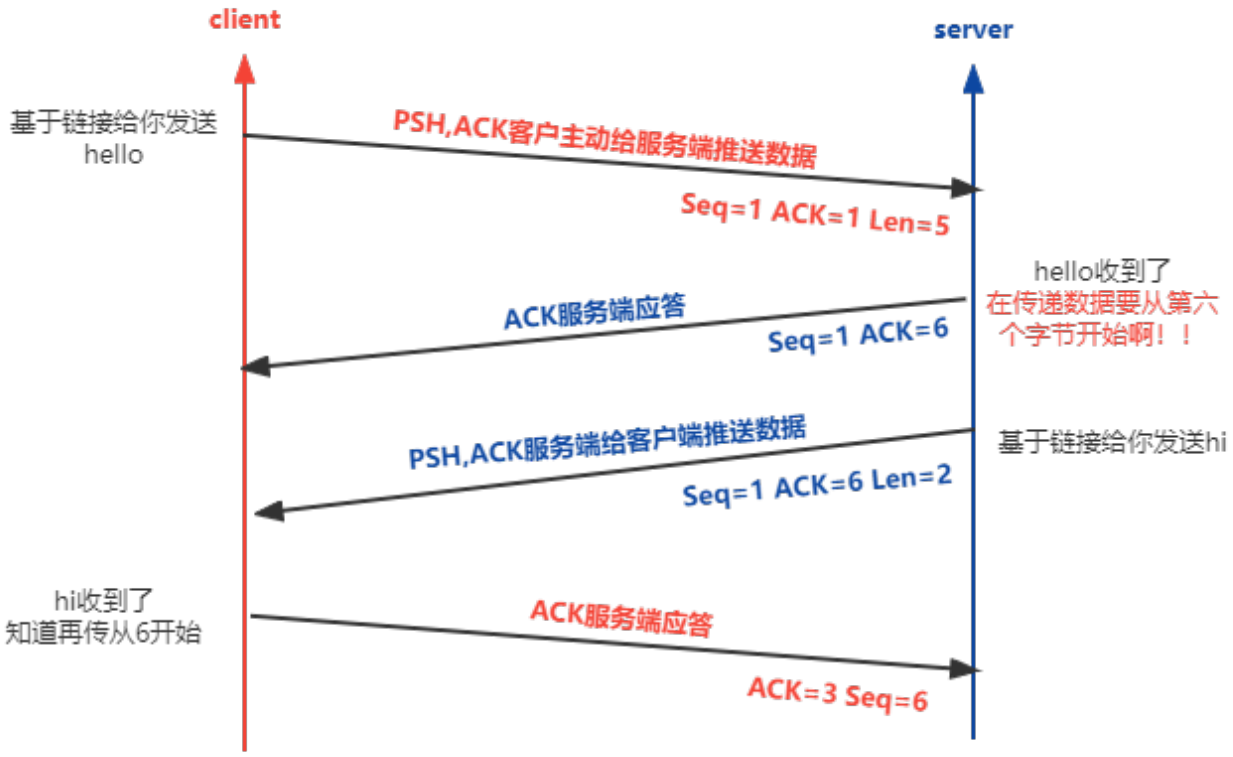
```
const net = require('net');
const server = net.createServer(function(socket)
{
    socket.on('data',function (data) { // 客户端
和服务端
        socket.write('hi'); // 服务端和客户端说 hi
    });
    socket.on('end',function () {
        console.log('客户端关闭')
    })
})
server.on('error',function(err){
    console.log(err);
})
server.listen(8080); // 监听8080端口
```

1）建立连接

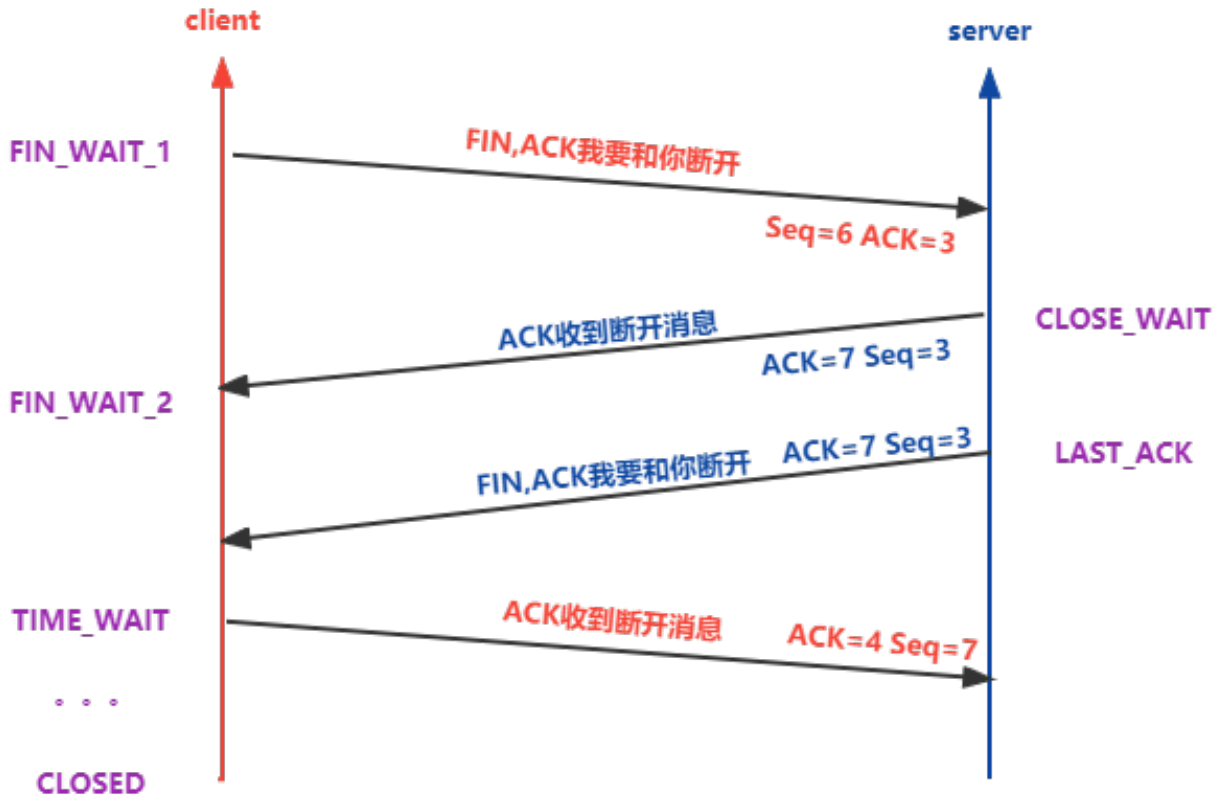


- 1）我能主动给你打·电话吗？ 2）当然可以啊！那我也能给你打电话吗?
- 3）可以的呢，建立连接成功!

2）数据传输


client
server
基于链接给你发送
hello
PSH,ACK客户主动给服务端推送数据
Seq=1 ACK=1 Len=5
hello收到了
在传递数据要从第六
个字节开始啊！！
ACK服务端应答
Seq=1 ACK=6
PSH,ACK服务端给客户端推送数据
基于链接给你发送hi
Seq=1 ACK=6 Len=2
hi收到了
知道再传从6开始
ACK服务端应答
ACK=3 Seq=6

3） 断开连接



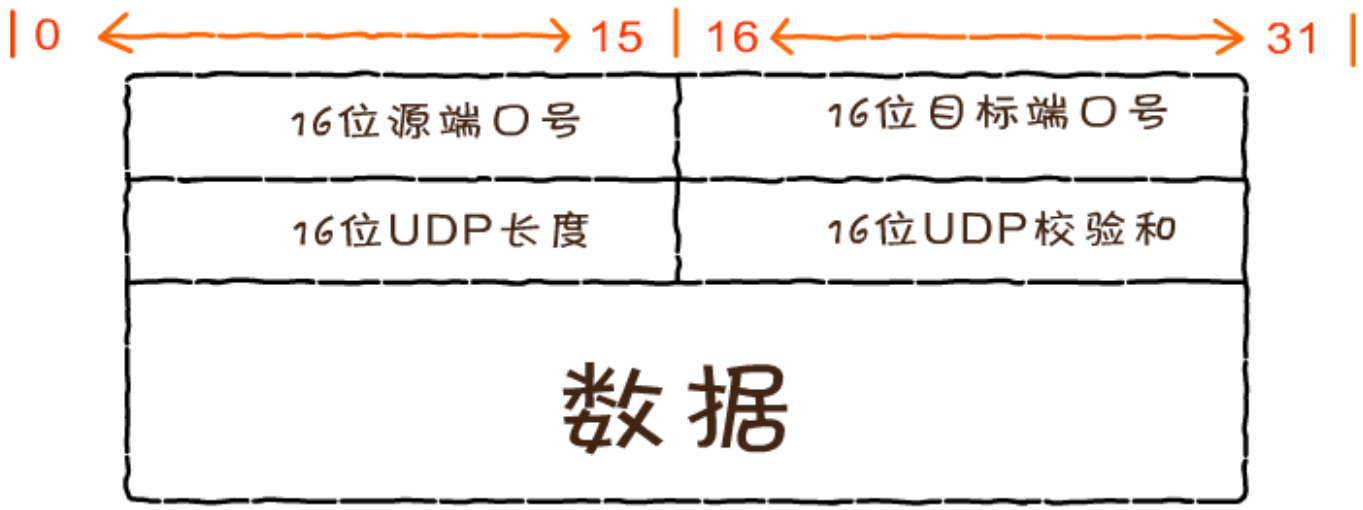
- 四次挥手
  - 1）我们分手吧 2）收到分手的信息
  - 3）好吧，分就分吧 4）行，那就到这里了

> 为了防止最终的`ACK`丢失，发送`ACK`后需要等待一段时间，因为如果丢包服务端需要重新发送`FIN`包，如果客户端已经`closed`，那么服务端会将结果解析成错误。 从而在高并发非长连接的场景下会有大量端口被占用。

3.UDP

`udp`用户数据报协议`User Datagram Protocol`，是一个无连接、不保证可靠性的传输层协议。你让我发什么就发什么！

- 使用场景：`DHCP`协议、`DNS`协议、`QUIC`协议等 (处理速度快，可以丢包的情况)



4.UDP抓包

server.js

```
var dgram = require("dgram");
var socket = dgram.createSocket("udp4");
socket.on("message", function (msg, rinfo) {
  console.log(msg.toString());
  console.log(rinfo);
  socket.send(msg, 0, msg.length, rinfo.port,
rinfo.address);
});
socket.bind(41234, "localhost");
```

client.js

```
var dgram = require('dgram');
var socket = dgram.createSocket('udp4');
socket.on('message',function(msg,rinfo){
    console.log(msg.toString());
    console.log(rinfo);
});
socket.send(Buffer.from('helloworld'),0,5,41234,
'localhost',function(err,bytes){
    console.log('发送了个%d字节',bytes);
});
socket.on('error',function(err){
    console.error(err);
});
```

> `udp.dstport`==41234

5.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，持续监测窗口大小）

6.粘包

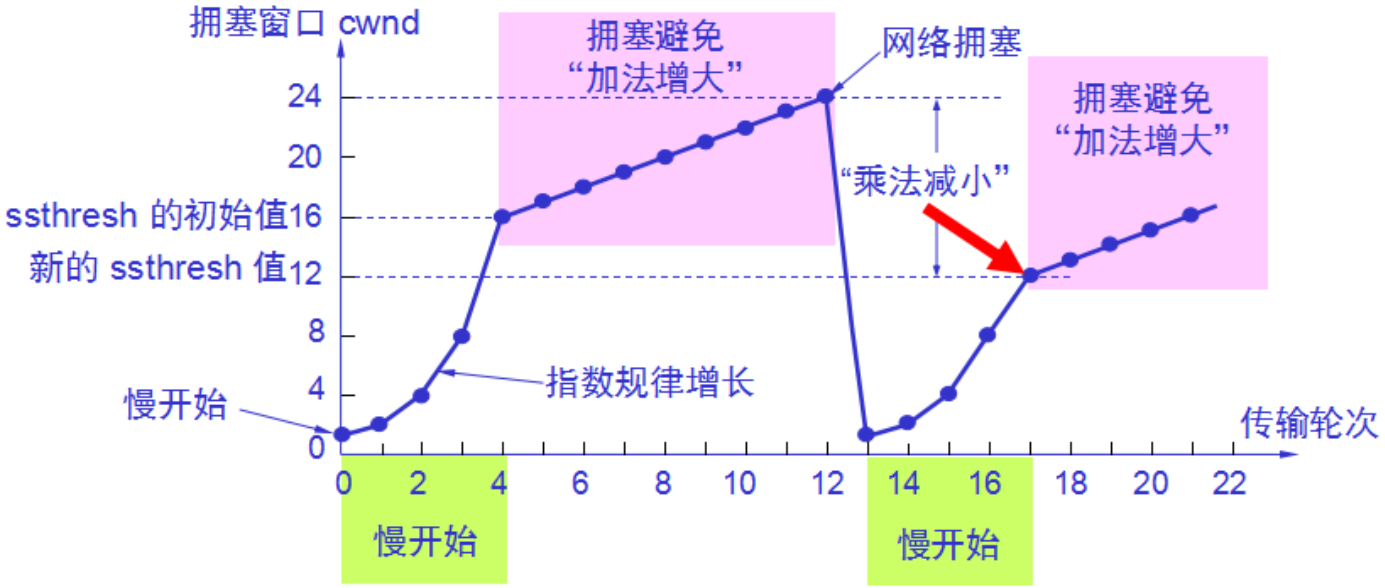
`Nagle`算法的基本定义是**任意时刻，最多只能有一个未被确认的小段** (TCP内部控制)

`Cork算法`当达到`MSS`(Maximum Segment Size)值时统一进行发送（此值就是帧的大小 - `ip`头 - `tcp`头 = 1460个字节）理论值

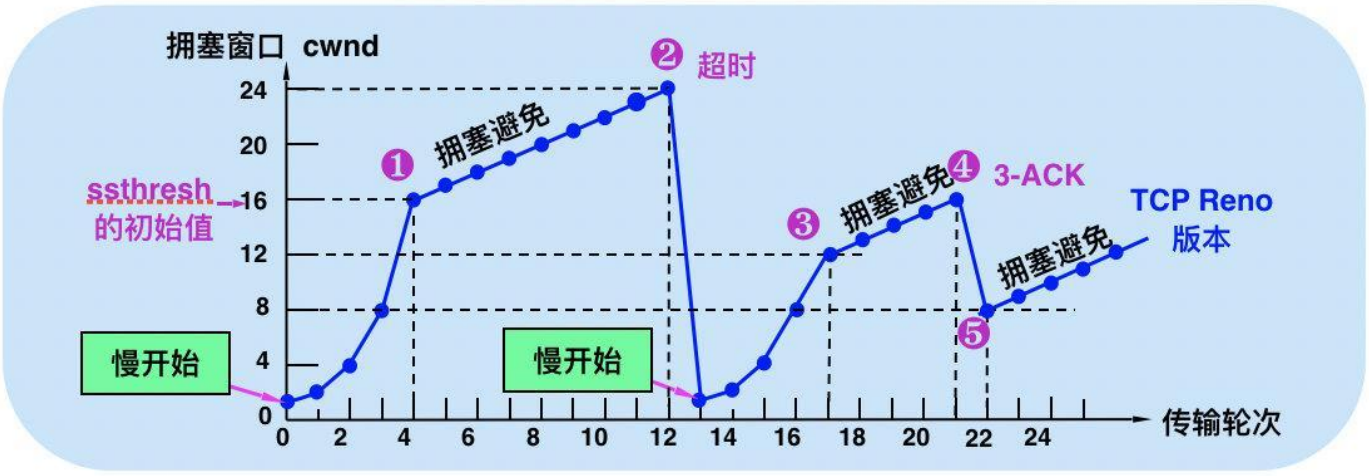
7.`TCP`拥塞处理

> 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，在传输过程中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



- 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。



> 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此，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。

HTTP

一.HTTP发展历程

1990年`HTTP/0.9`为了便于服务器和客户端处理，采用了“纯文本”格式，只运行使用GET请求。在响应请求之后会立即关闭连接。

1996年`HTTP/1.0`增强了 0.9 版本，引入了 HTTP Header（头部）的概念，传输的数据不再仅限于文本，可以解析图片音乐等，增加了响应状态码和`POST`,`HEAD`等请求方法。（内容协商）

1999年广泛使用`HTTP/1.1`，正式标准，允许持久连接，允许响应数据分块，增加了缓存管理和控制，增加了 PUT、DELETE 等新的方法。(问题 多个请求并发 http队头阻塞的问题)

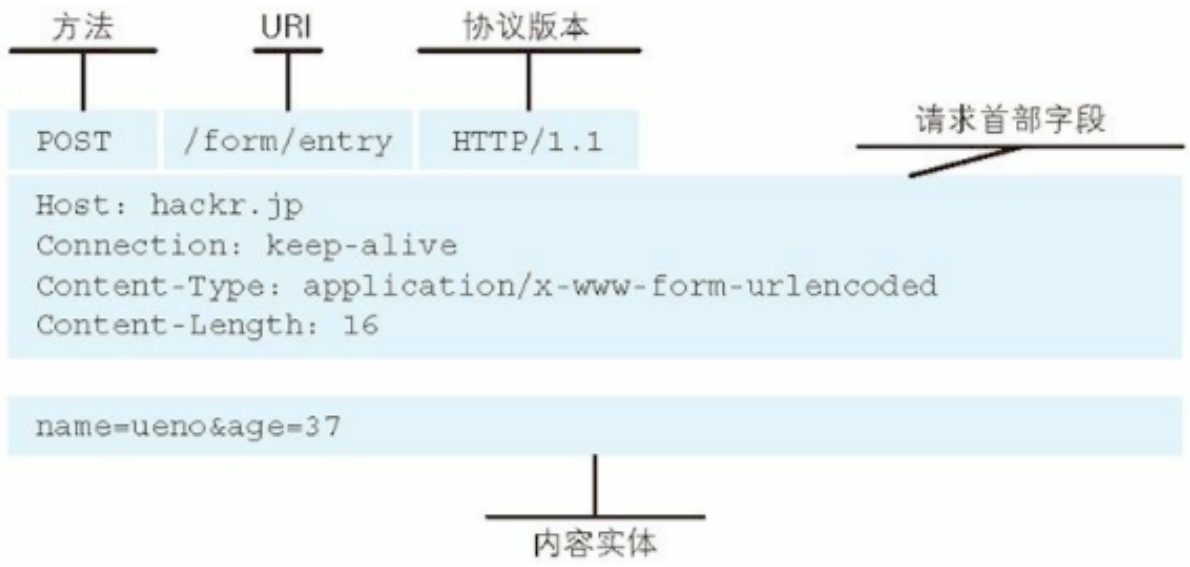
2015年`HTTP/2`，使用`HPACK`算法压缩头部，减少数据传输量。允许服务器主动向客户端推送数据，二进制协议可发起多个请求，使用时需要对请求加密通信。

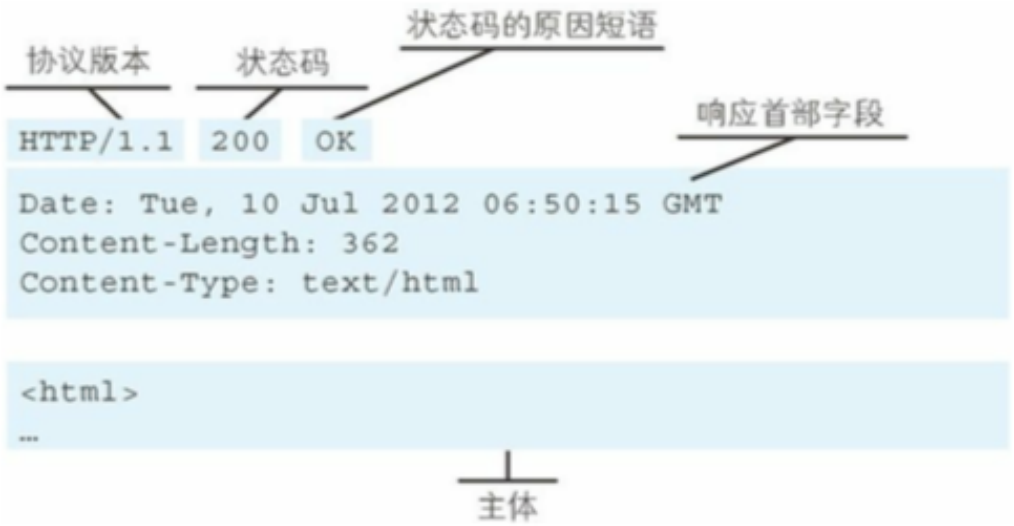
2018年`HTTP/3` 基于`UDP`的`QUIC`协议。

二.HTTP/1.1

- HTTP/1.1 是可靠传输协议，基于 `TCP/IP` 协议；
- 采用应答模式，客户端主动发起请求，服务器被动回复请求；

- HTTP是无状态的每个请求都是互相独立
- HTTP 协议的请求报文和响应报文的结构基本相同，由三部分组成。





三.HTTP/1.1特点

1.长连接

`TCP` 的连接和关闭非常耗时间，所以我们可以复用`TCP`创建的连接。HTTP/1.1响应中默认会增加`Connection:keep-alive`

2.管线化

在同一条`TCP`连接来进行数据的收发，就会变成 "串行" 模式，如果某个请求过慢就会发生阻塞问题。 ***Head-of-line blocking***
管线化就是在同一个TCP连接中同时发送多个HTTP请求。

> 默认浏览器不开启管线化。

3.`Cookie`

Set-Cookie/Cookie用户第一次访问服务器的时候，服务器会写入身份标识，下次再请求的时候会携带`cookie`。通过Cookie可以实现有状态的会话

4.内容协商

客户端和服务端进行协商，返回对应的结果

客户端 Header	服务端 Header	
Accept	Content-Type	我发送给你的数据是什么类型
Accept-encoding	Content-Encoding	我发送给你的数据是用什么格式压缩（`gzip`、`deflate`、`br`）
Accept-language		根据客户端支持的语言返回 （多语言）
Range	Content-Range	范围请求数据 206

5.`HTTP`缓存

强缓存 服务器会将数据和缓存规则一并返回，缓存规则信息包含在响应header中。 `Cache-Control`

对比缓存 **`if-Modified-Since/if-None-Match`**（最后修改时间）、**`Last-modified/Etag`**(指纹)