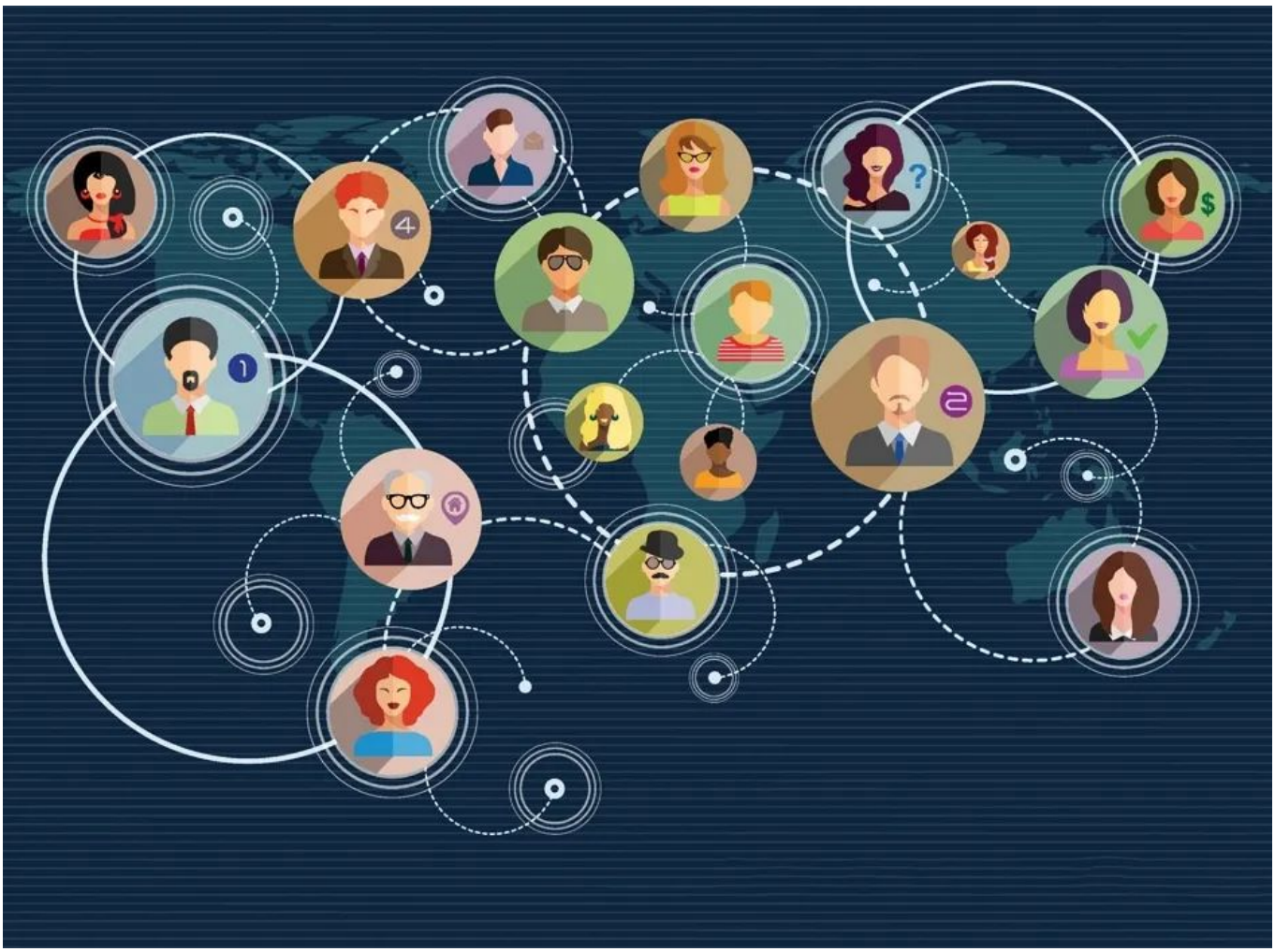
「计算机网络」夺命连环12问

CodeSheep 2021-03-22 09:36

以下文章来源于艾小仙 ， 作者艾小仙

艾小仙

一个愤世嫉俗，脱离低级趣味的人



往期肝货笔记整理

- 数据结构和算法刷题笔记.pdf下载
- 找工作简历模板集(word格式)下载
- Java基础核心知识大总结.pdf 下载
- C/C++常见面试题（含答案）下载
- 设计模式学习笔记.pdf下载
- Java后端开发学习路线+知识点总结
- 前端开发学习路线+知识点总结

- 大数据开发学习路线+知识点总结
- C/C++(后台)学习路线+知识点总结
- 嵌入式开发学习路线+知识点总结

谈一谈你对TCP/IP四层模型，OSI七层模型的理解?

为了增强通用性和兼容性，计算机网络都被设计成层次机构，每一层都遵守一定的规则。

因此有了OSI这样一个抽象的网络通信参考模型，按照这个标准使计算机网络系统可以互相连接。

物理层：通过网线、光缆等这种物理方式将电脑连接起来。传递的数据是**比特流**，0101010100。

数据链路层：首先，把比特流封装成数据**帧**的格式，对0、1进行分组。电脑连接起来之后，数据都经过网卡来传输，而网卡上定义了全世界唯一的MAC地址。然后再通过广播的形式向局域网内所有电脑发送数据，再根据数据中MAC地址和自身对比判断是否是发给自己的。

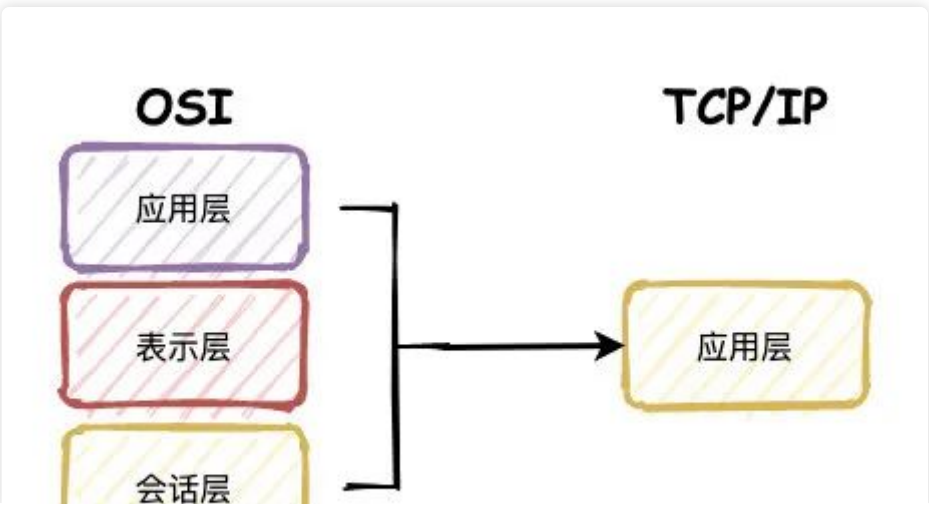
网络层：广播的形式太低效，为了区分哪些MAC地址属于同一个子网，网络层定义了IP和子网掩码，通过对IP和子网掩码进行与运算就知道是否是同一个子网，再通过路由器和交换机进行传输。IP协议属于网络层的协议。

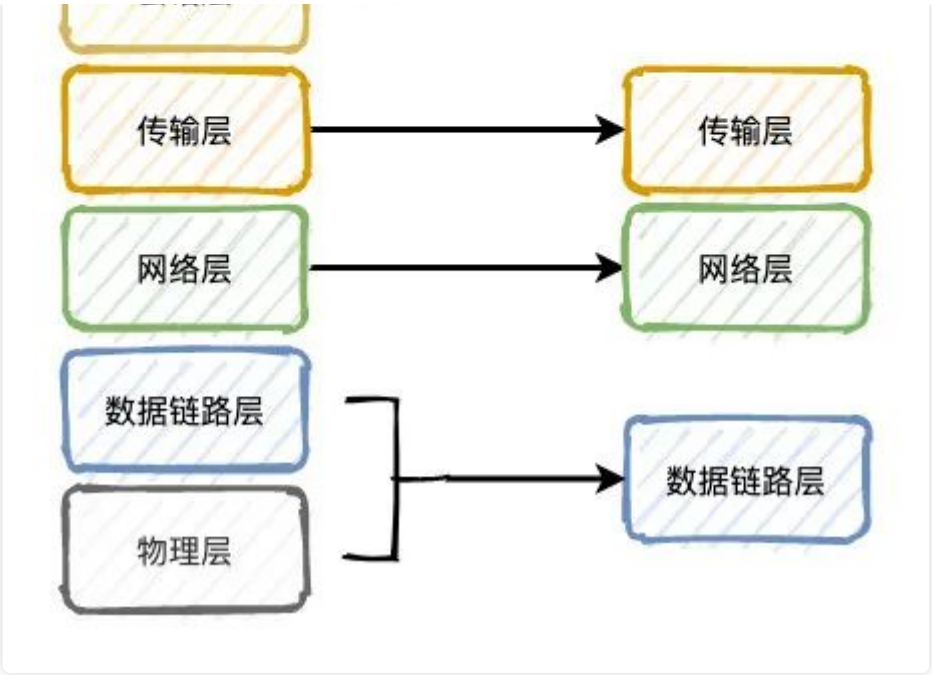
传输层：有了网络层的MAC+IP地址之后，为了确定数据包是从哪个进程发送过来的，就需要端口号，通过端口来建立通信，比如TCP和UDP属于这一层的协议。

会话层：负责建立和断开连接

表示层：为了使得数据能够被其他的计算机理解，再次将数据转换成另外一种格式，比如文字、视频、图片等。

应用层：最高层，面对用户，提供计算机网络与最终呈现给用户的界面





TCP/IP则是四层的结构，相当于是对OSI模型的简化。

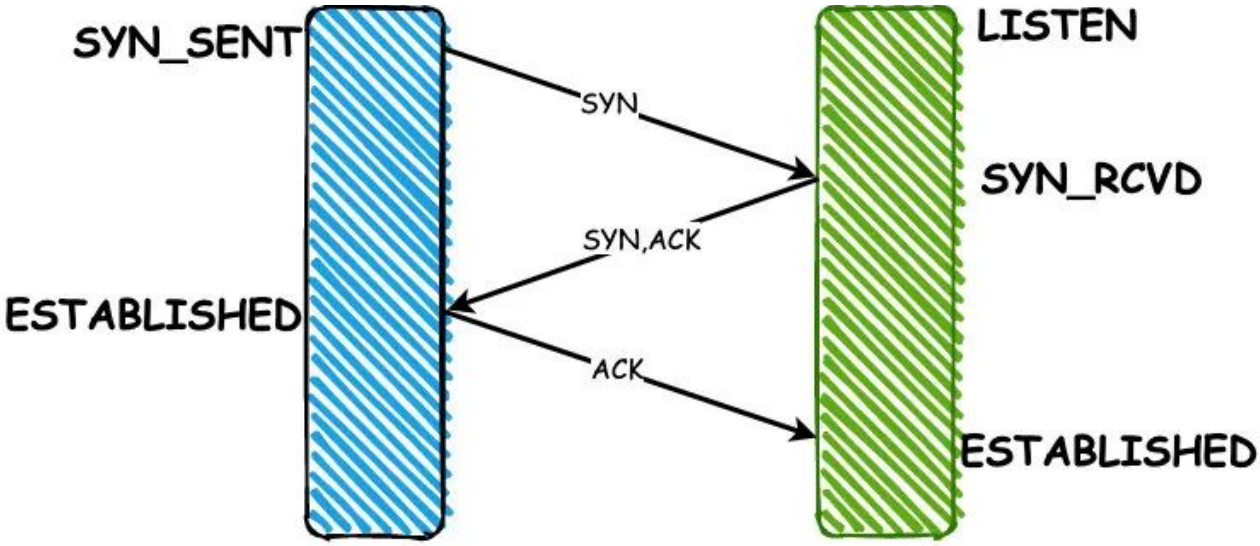
1. 数据链路层，也有称作网络访问层、网络接口层。他包含了OSI模型的物理层和数据链路层，把电脑连接起来。
2. 网络层，也叫做IP层，处理IP数据包的传输、路由，建立主机间的通信。
3. 传输层，就是为两台主机设备提供端到端的通信。
4. 应用层，包含OSI的会话层、表示层和应用层，提供了一些常用的协议规范，比如FTP、SMPT、HTTP等。

总结下来，就是物理层通过物理手段把电脑连接起来，数据链路层则对比特流的数据进行分组，网络层来建立主机到主机的通信，传输层建立端口到端口的通信，应用层最终负责建立连接，数据格式转换，最终呈现给用户。

说说TCP 3次握手的过程?

建立连接前server端需要监听端口，所以初始状态是LISTEN。

1. client端建立连接，发送一个SYN同步包，发送之后状态变成SYN_SENT
2. server端收到SYN之后，同意建立连接，返回一个ACK响应，同时也会给client发送一个SYN包，发送完成之后状态变为SYN_RCVD
3. client端收到server的ACK之后，状态变为ESTABLISHED，返回ACK给server端。server收到之后状态也变为ESTABLISHED，连接建立完成。



为什么要3次？2次，4次不行吗？

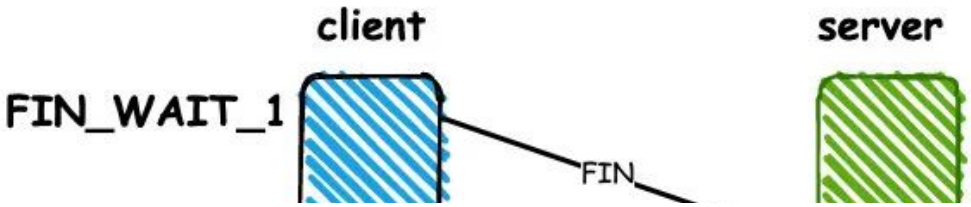
因为TCP是双工传输模式，不区分客户端和服务端，连接的建立是双向的过程。

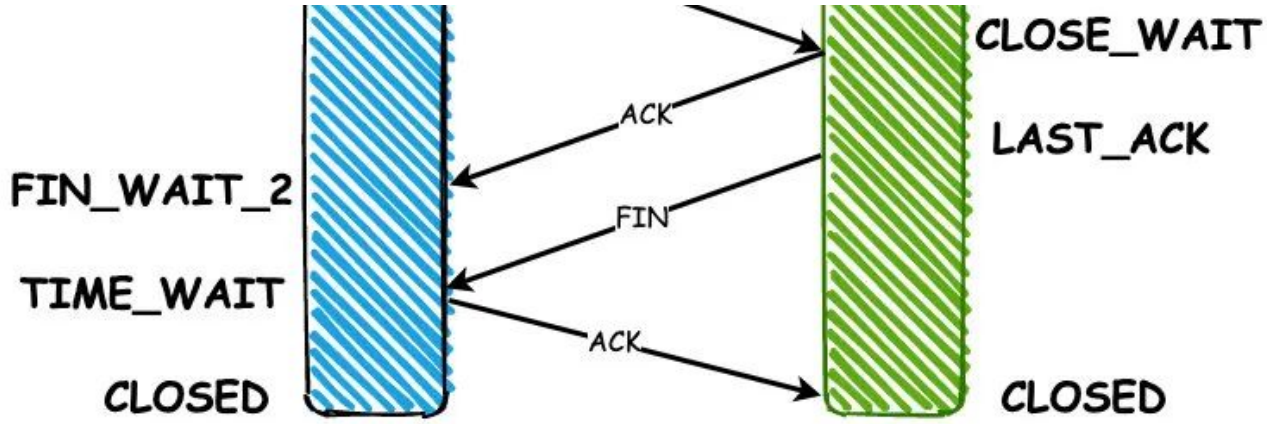
如果只有两次，无法做到双向连接的建立，从建立连接server回复的SYN和ACK合并成一次可以看出来，他也不需要4次。

挥手为什么要四次？因为挥手的ACK和FIN不能同时发送，因为数据发送的截止时间不同。

那么四次挥手的过程呢？

1. client端向server发送FIN包，进入FIN_WAIT_1状态，这代表client端已经没有数据要发送了
2. server端收到之后，返回一个ACK，进入CLOSE_WAIT等待关闭的状态，因为server端可能还有没有发送完成的数据
3. 等到server端数据都发送完毕之后，server端就向client发送FIN，进入LAST_ACK状态
4. client收到ACK之后，进入TIME_WAIT的状态，同时回复ACK，server收到之后直接进入CLOSED状态，连接关闭。但是client要等待2MSL(报文最大生存时间)的时间，才会进入CLOSED状态。





为什么要等待2MSL的时间才关闭?

1. 为了保证连接的可靠关闭。如果server没有收到最后一个ACK，那么就会重发FIN。
2. 为了避免端口重用带来的数据混淆。如果client直接进入CLOSED状态，又用相同端口号向server建立一个连接，上一次连接的部分数据在网络中延迟到达server，数据就可能发生混淆了。

TCP怎么保证传输过程的可靠性?

校验和：发送方在发送数据之前计算校验和，接收方收到数据后同样计算，如果不一致，那么传输有误。

确认应答，序列号：TCP进行传输时数据都进行了编号，每次接收方返回ACK都有确认序列号。

超时重传：如果发送方发送数据一段时间后没有收到ACK，那么就重发数据。

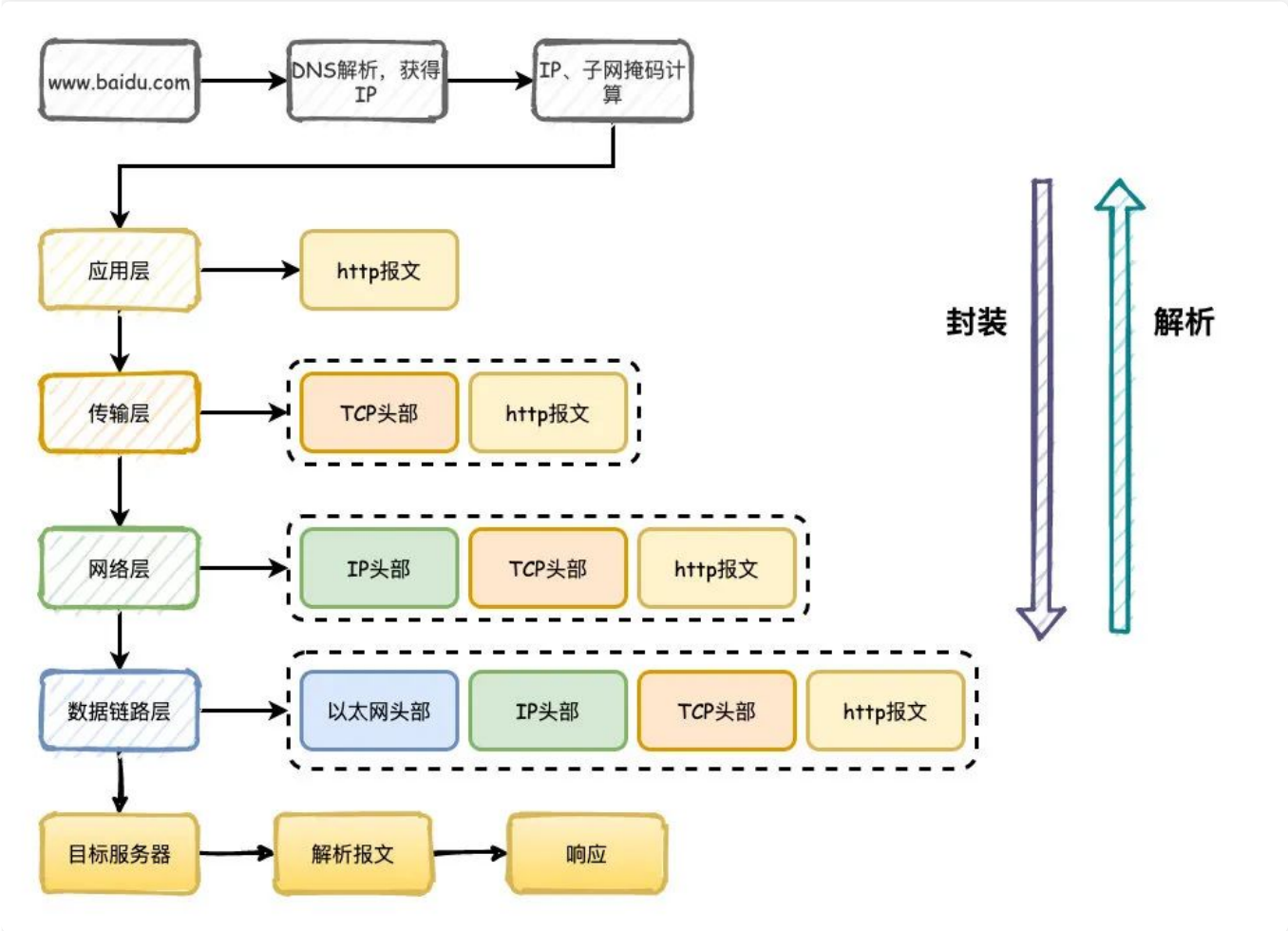
连接管理：三次握手和四次挥手的过程。

流量控制：TCP协议报头包含16位的窗口大小，接收方会在返回ACK时同时把自己的即时窗口填入，发送方就根据报文中窗口的大小控制发送速度。

拥塞控制：刚开始发送数据的时候，拥塞窗口是1，以后每次收到ACK，则拥塞窗口+1，然后将拥塞窗口和收到的窗口取较小值作为实际发送的窗口，如果发生超时重传，拥塞窗口重置为1。这样做的目的就是为了保证传输过程的高效性和可靠性。

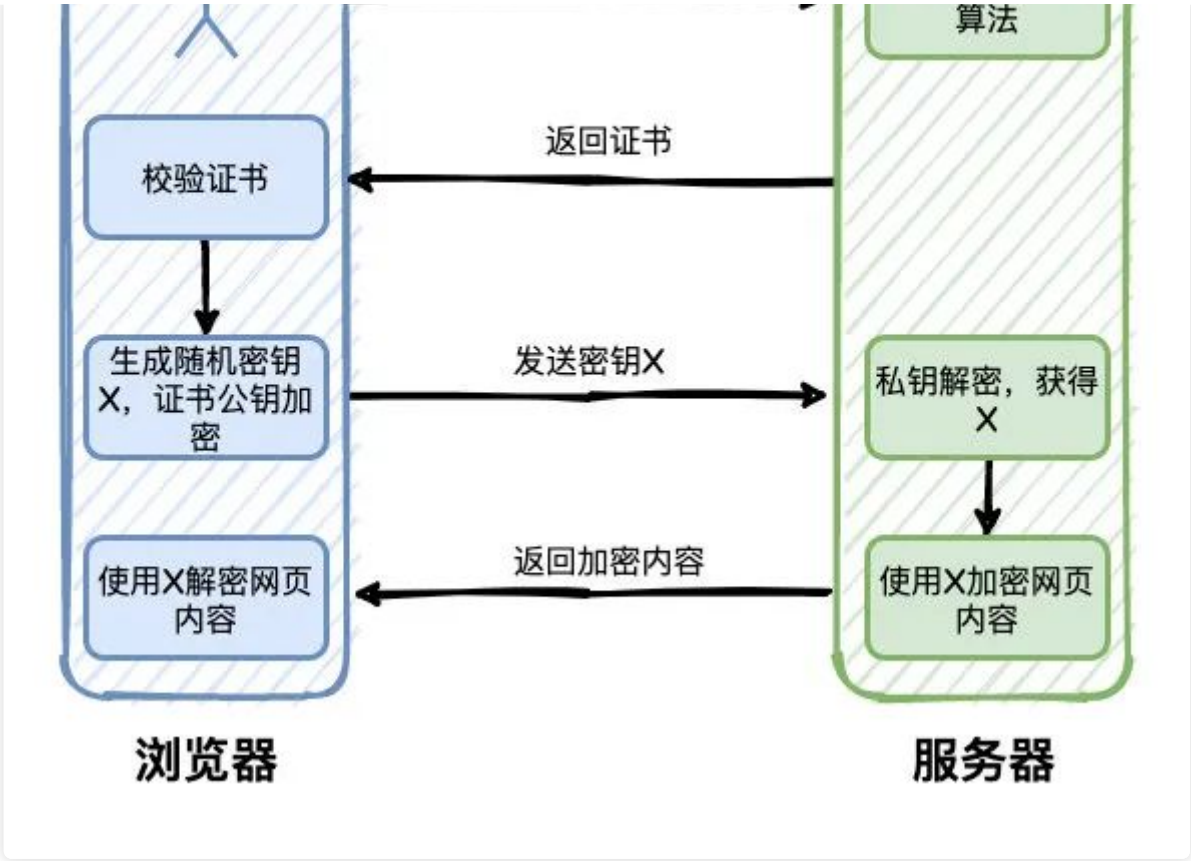
说下浏览器请求一个网址的过程?

1. 首先通过DNS服务器把域名解析成IP地址，通过IP和子网掩码判断是否属于同一个子网
2. 构造应用层请求http报文，传输层添加TCP/UDP头部，网络层添加IP头部，数据链路层添加以太网协议头部
3. 数据经过路由器、交换机转发，最终达到目标服务器，目标服务器同样解析数据，最终拿到http报文，按照对应的程序的逻辑响应回去。



知道HTTPS的工作原理吗?

1. 用户通过浏览器请求https网站，服务器收到请求，选择浏览器支持的加密和hash算法，同时返回数字证书给浏览器，包含颁发机构、网址、公钥、证书有效期等信息。
2. 浏览器对证书的内容进行校验，如果有问题，则会有一个提示警告。否则，就生成一个随机数X，同时使用证书中的公钥进行加密，并且发送给服务器。
3. 服务器收到之后，使用私钥解密，得到随机数X，然后使用X对网页内容进行加密，返回给浏览器
4. 浏览器则使用X和之前约定的加密算法进行解密，得到最终的网页内容



负载均衡有哪些实现方式?

DNS：这是最简单的负载均衡的方式，一般用于实现地理级别的负载均衡，不同地域的用户通过DNS的解析可以返回不同的IP地址，这种方式的负载均衡简单，但是扩展性太差，控制权在域名服务商。

Http重定向：通过修改Http响应头的Location达到负载均衡的目的，Http的302重定向。这种方式对性能有影响，而且增加请求耗时。

反向代理：作用于应用层的模式，也被称作为**七层负载均衡**，比如常见的Nginx，性能一般可以达到万级。这种方式部署简单，成本低，而且容易扩展。

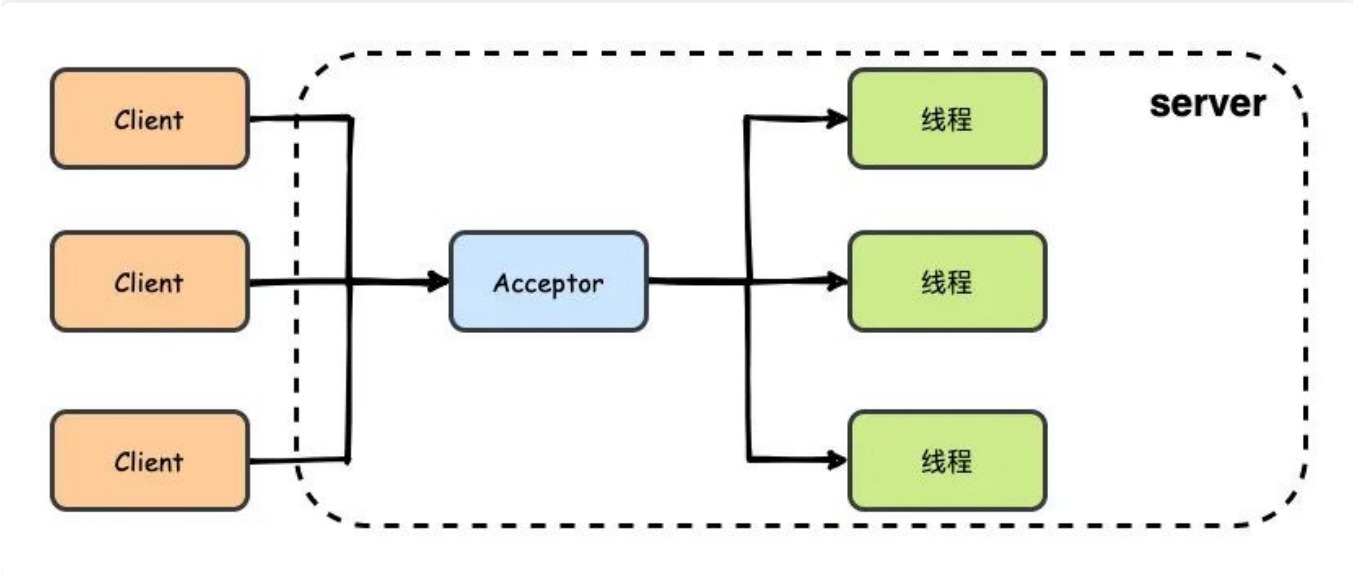
IP：作用于网络层的和传输层的模式，也被称作**四层负载均衡**，通过对数据包的IP地址和端口进行修改来达到负载均衡的效果。常见的有LVS（Linux Virtual Server），通常性能可以支持10万级并发。

按照类型来划分的话，还可以分成DNS负载均衡、硬件负载均衡、软件负载均衡。

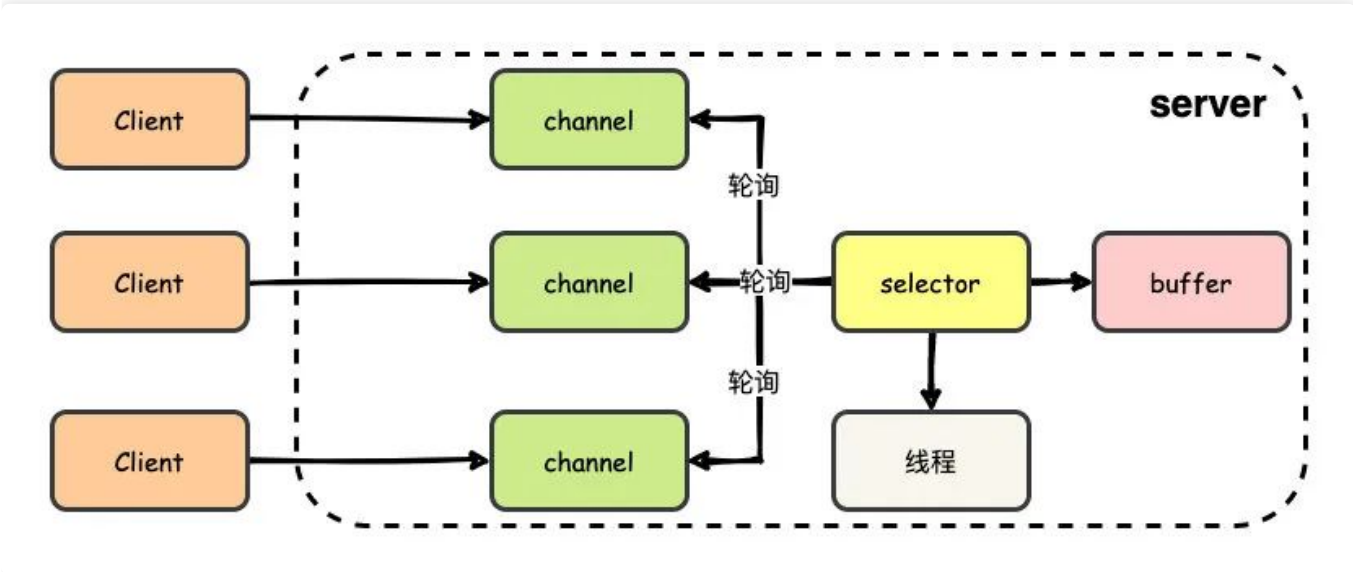
其中硬件负载均衡价格昂贵，性能最好，能达到百万级，软件负载均衡包括Nginx、LVS这种。

说说BIO/NIO/AIO的区别?

BIO：同步阻塞IO，每一个客户端连接，服务端都会对应一个处理线程，对于没有分配到处理线程的连接就会被阻塞或者拒绝。相当于是**一个连接一个线程**。



NIO：同步非阻塞IO，基于Reactor模型，客户端和channel进行通信，channel可以进行读写操作，通过多路复用器selector来轮询注册在其上的channel，而后再进行IO操作。这样的话，在进行IO操作的时候再用一个线程去处理就可以了，也就是**一个请求一个线程**。



AIO：异步非阻塞IO，相比NIO更进一步，完全由操作系统来完成请求的处理，然后通知服务端开启线程去进行处理，因此是**一个有效请求一个线程**。

那么你怎么理解同步和阻塞?

首先，可以认为一个IO操作包含两个部分：

1. 发起IO请求
2. 实际的IO读写操作

同步和异步在于第二个，实际的IO读写操作，如果操作系统帮你完成了再通知你，那就是异步，否则都叫做同步。

阻塞和非阻塞在于第一个，发起IO请求，对于NIO来说通过channel发起IO操作请求后，其实就返回了，所以是非阻塞。

谈一下你对Reactor模型的理解?

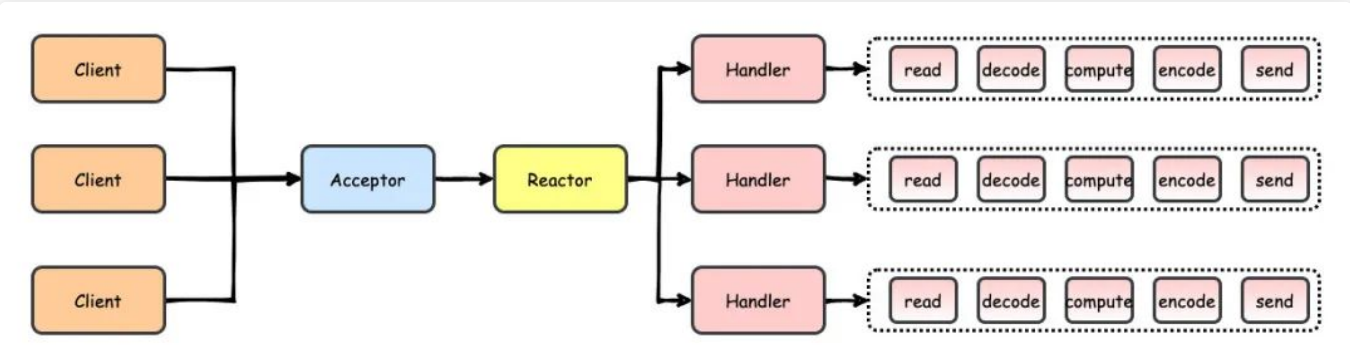
Reactor模型包含两个组件：

1. Reactor：负责查询、响应IO事件，当检测到IO事件时，分发给Handlers处理。
2. Handler：与IO事件绑定，负责IO事件的处理。

它包含几种实现方式：

单线程Reactor

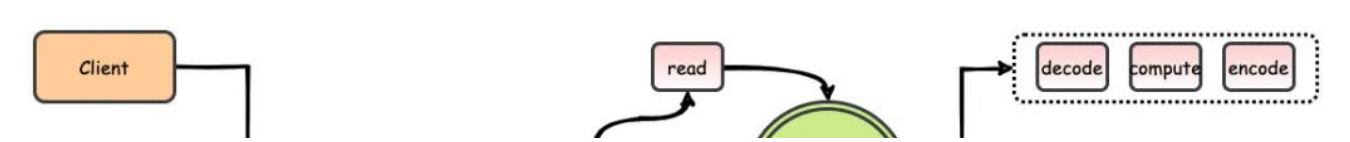
这个模式reactor和handler在一个线程中，如果某个handler阻塞的话，会导致其他所有的handler无法执行，而且无法充分利用多核的性能。

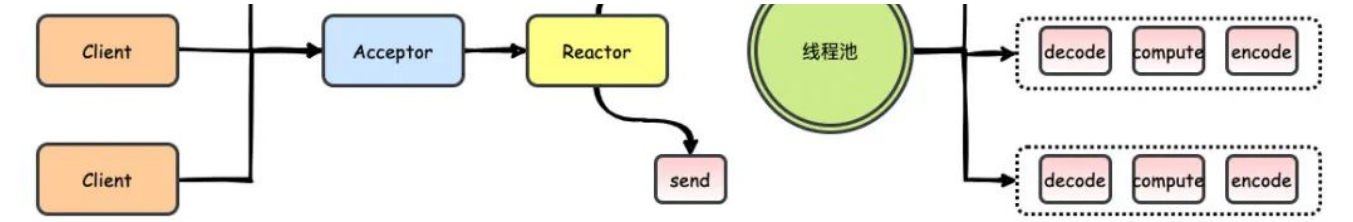


单Reactor多线程

由于decode、compute、encode的操作并非IO的操作，多线程Reactor的思路就是充分发挥多核的特性，同时把非IO的操作剥离开。

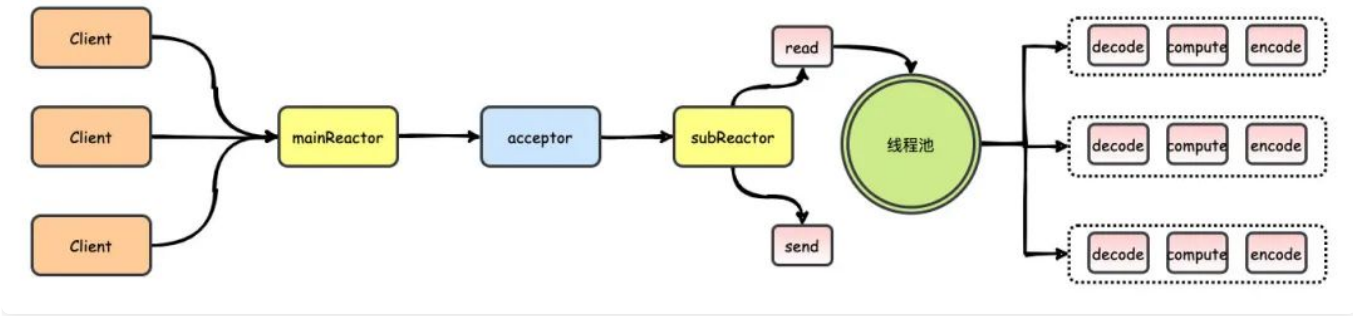
但是，单个Reactor承担了所有的事件监听、响应工作，如果连接过多，还是可能存在性能问题。





多Reactor多线程

为了解决单Reactor的性能问题，就产生了多Reactor的模式。其中mainReactor建立连接，多个subReactor则负责数据读写。



····· END ·····

往期肝货笔记整理

- 数据结构和算法刷题笔记.pdf下载
- 找工作简历模板大分享.doc下载
- Java基础核心知识大总结.pdf 下载
- C/C++常见面试题（含答案）下载
- 设计模式学习笔记.pdf下载
- Java后端开发学习路线+知识点总结
- 前端开发学习路线+知识点总结
- 大数据开发学习路线+知识点总结
- C/C++(后台)学习路线+知识点总结
- 嵌入式开发学习路线+知识点总结