**三次握手 + 四次挥手机制**

**三次握手：**

三次握手是建立TCP连接的建立过程。客户端结束CLOSE阶段

服务端结束CLOSE端。 服务端处于LISTEN阶段

1. 首先客户端向服务器发送SYN包 等待服务器确认

2. 服务端接收到客户端发来的SYN包 服务端结束LISTEN阶段 并返回TCP报文

其中包含SYN和ACK

3. 客户端接收到发送的SYN + ACK包后 明确了客户端到服务器的数据传输是正常的 从而结束SYN-SENT阶段

并返回最后一段报文 包含ACK包和序列号x+1 随后客户端进入ESTABLISHED阶段

4. 当服务器收到来自客户端的确认报文之后 得知服务器到客户端的数据传输是正常的 结束SYN-RECV

阶段 进入ESTABLISHED阶段

**四次挥手：**

TCP连接的释放。客户端主动释放连接 结束ESTABLISHED阶段

1. 首先客户端向服务器发送TCP报文请求释放TCP连接 之后进入半关闭状态

2. 服务器接收到客户端请求的 FIN报文 结束ESTABLISHED阶段 进入CLOSE-WAIT阶段 并返回ACK报文

3. 服务端在发出ACK确认报文后 服务器将遗留的数据传送给客户端 待传输完成后再向客户端发出FIN和ACK报文

4. 客户端收到从服务器发来的TCP报文 确认了服务器已经准备好释放连接 进入TIME-WAIT阶段 并向服务器发送ACK报文

随后客户端等待2个MSL。进入CLOSE阶段

**为什么要进行三次握手？两次握手行不行？**

三次握手的主要目的是确认自己和对方的发送和接收都是正常的 从而保证了双方的可靠通信 实现全双工通信。

如果只有两次握手 即只有客户端直到服务器能够正常接收到自己的信息 但是服务器不知道服务端能否接收到自己的信息

如果服务端发送给客户端的包丢了 服务端就建立TCP连接 一直傻等 会造成阻塞

**如果三次握手的时候每次握手信息对方没有收到怎么办？**

第一次握手服务器未接收到客户端的SYN报文：等待一段时间后重新发送SYN同步报文 达到最大重传限制 系统调用会返回-1

第二次握手客户端没有接收到服务端的ACK报文：客户端采取第一次握手失败的动作。

第三次握手服务器未接受到服务端发送的ACK报文：采用超时重传机制

第二次握手传回了ACK 为什么要传回SYN

ACK是为了告诉客户端发来的数据接收无误 而SYN是服务端的连接请求

**为什么要四次挥手？**

当主动方在数据传送结束后发出连接释放的通知，由于被动方可能还有必要的数据要处理，所以会先返回 ACK 确认收到报文. 当被动方也没有数据再发送的时候，则发出连接释放通知，对方确认后才完全关闭TCP连接。简单来说就是因为TCP是全双工的，两个方向的连接需要单独关闭。

**CLOSE-WAIT 和 TIME-WAIT 的状态和意义**

**CLOSE-WAIT**：发生在第二次挥手 当服务器收到了客户端的关闭请求 进入CLSOE-WAIT阶段 然而服务端还有些数据没传完所以不能立即关闭连接 CLOSE-WAIT状态是保证服务器发送未发生的数据。

**TIME-WAIT**： 发生在第四次挥手 是客户端必需等待的时间 目的是等待服务端的关闭与确保客户端发送的ACK报文能够成功到达 ； 如果不存在这个步骤会导致 ： 1. 下一次建立TCP连接时 上次的残留数据包会引起异常 2.如果服务端超时重传FIN包 会报异常 但是其实没有问题的

**TIME\_WAIT 状态会导致什么问题，怎么解决**

在高并发短连接的场景下 服务器会有大量的连接处于TIME-WAIT状态 每个连接会占有文件描述符 文件描述符是有上限的, 如果持续高并发 会导致正常的连接失败。

**TIME-WAIT 为什么是 2MSL**

因为客户端发出最后的ACK确认报文 并不能确定服务端能接收到ACK报文 所以会设置一个时长2MSL的计时器

如果服务器在1MSL内没有收到客户端发出的ACK确认报文 则重传FIN报文

如果客户端在2MSL内没有再次受到服务器发送的FIN报文 说明正常接收到了ACK报文 进入CLOSE阶段

所以目的就是确认服务器接收到客户端最后发出的ACK确认报文。

**TCP 和 UDP 的区别**

TCP：面向连接 可靠传输 采用字节流 传输效率慢 所需资源多 一般用于文件传输和邮件传输

UDP：无面向连接 不可靠传输 采用数据包文段的形式传输 传输效率快 所需资源少 一般用于即时通信。

**TCP 是如何保证可靠性的**

**数据分块:** 应用数据被分割成 TCP 认为最适合发送的数据块。

**校验和** TCP 将保持它首部和数据部分的检验和。

**流量控制** TCP 连接的双方都有一个固定大小的缓冲空间，发送方发送的数据量不能超过接收端缓冲区的大小

**拥塞控制** 当网络某个节点发生拥塞时，减少数据的发送。

**超时重传** 当 TCP 发出一个报文段后，它启动一个定时器，等待目的端确认收到这个报文段。如果超过某个时间还没有收到确认，将重发这个报文段。

**UDP 为什么是不可靠的？bind 和 connect 对于 UDP 的作用是什么**

UDP只有一个socket接收缓冲区 没有socket发送缓冲区 所以有数据就发 不管对方是否能够正确接收。当对方socket接收缓冲区满了之后 新来的数据无法进入缓冲 此数据报就会被丢弃，因此 UDP 不能保证数据能够到达目的地，此外，UDP 也没有流量控制和重传机制，故UDP的数据传输是不可靠的。

**TCP 超时重传的原理**

发送方发送数据后会设置一个定时器 如果一定时间内没有收到数据包的ACK报文 那么就重新发送数据 达到一定次数还没有成功的话就放弃重传并发送一个复位信号。

**TCP 的停止等待协议是什么**

为了实现TCP可靠传输而提供的相对简单的协议

该协议指的是发送方发完一组数据之后 直到接收到接收方的ACK报文才继续传输下一组数据。

**TCP 流量控制与拥塞控制**

流量控制：让发送方的发送速率不要太快 让接收方来得及接收。通过滑动窗口机制可以很方便的实现。

拥塞控制：防止过多的数据注入网络中 避免出现网络负载过大的情况。

区别：拥塞控制是全局的 流量控制往往是针对局部的点对点通信。

**如果接收方滑动窗口满了，发送方会怎么做**

TCP流量控制的滑动窗口协议 接收方返回给发送方的ACK报文里会包含接收窗口的大小

如果接收窗口已满 接收方会给发送方的接收窗口大小为0 发送方则等待接收方的窗口大小直到变为非0为止

接收方回应的ACK报文会丢失 防止双方互相等待出现死锁 计时器来辅助周期性查询。

**TCP 拥塞控制采用的四种算法**

**慢开始:**

一开始发送少量的数据先探测一下网络状况，即由小到大的增大发送窗口（拥塞窗口 cwnd）

**拥塞避免**

拥塞控制是为了让拥塞窗口 cwnd 缓慢地增大，即每经过一个往返时间 RTT （往返时间定义为发送方发送数据到收到确认报文所经历的时间）就把发送方的 cwnd 值加 1，通过让 cwnd 线性增长，防止很快就遇到网络拥塞状态。

**快重传**

快重传算法要求接收方每收到一个失序的报文就立即发送重复确认，而不要等到自己发送数据时才捎带进行确认，假定发送方发送了 Msg 1 ~ Msg 4 这 4 个报文，已知接收方收到了 Msg 1，Msg 3 和 Msg 4 报文，此时因为接收到收到了失序的数据包，按照快重传的约定，接收方应立即向发送方发送 Msg 1 的重复确认

**快恢复** 快恢复算法是和快重传算法配合使用的，该算法主要有以下两个要点：

① 当发送方连续收到三个重复确认，执行乘法减小，慢开始门限 ssthresh 值减半；

② 由于发送方可能认为网络现在没有拥塞，因此与慢开始不同，把 cwnd 值设置为 ssthresh 减半之后的值，然后执行拥塞避免算法，线性增大 cwnd。

**TCP 粘包问题**

发送方数据发送太空 接收方赶不上发送端的速度 发生粘包

设置分隔符 从数据流分离出消息内容 比如换行符

**SYN FLOOD 是什么**

DoS攻击。 消耗服务器所有可用资源使服务器无法用于处理合法请求。

**为什么服务端易受到 SYN 攻击**

TCP建立过程中 服务端不确定自己发送给客户端的SYN-ACK消息或客户端反馈的ACK报文会丢在半路

所以会在半连接状态设置定时器 服务端为了维持半连接状态 需要分配内核资源维护 攻击者伪造大量虚假IP向

服务端发送SYN包 形成SYN FLOOD攻击 攻击者故意不响应ACK消息 导致服务器大量注入不能完成的半连接占据 直到

资源耗尽。

解决方案： SYN cache： 用于辨别恶意IP的路由器。