## linux 部分

1. linux当在一个目录下创建文件时，无法创建，可能是什么问题？

①磁盘空间满了

②inode节点不足

③系统ulimit限制打开文件数目

1. ext2、ext3、ext4文件系统的区别？

ext2和ext3的区别就是一个磁盘日志功能。

ext3与ext4:的区别：

①支持更大容量的文件系统和文件，ext3支持16TB和最大2T把文件，ext4支持1EB和最大16TB文件

②Extends，操作大文件效率要高

③多块一次性分配ext3每次比如分配4KB的page来写，ext4可以一次多个page写

④延时分配

⑤快速fsck

⑥日志校验

⑦无日志模式

⑧在线碎片整理

⑨默认启用barrier，实际上就是缓存和日志的关系，写入斗鱼缓存，只有当日志记录之后才进行写磁盘操作。

1. linux一些用于监控的命令？

①w

查看系统负载，当前有哪些用户登录到系统中以及一些行为

登录用户 终端号 fromip 登录时间 1 5 15分钟平均负载

②vmstat

比较好用，可以看正在使用cpu的进程数、阻塞进程数、内存空间、磁盘IO、cpu空闲使用情况

③top

主要是cpu和内存的使用情况

④sar

一般用来查看各个网卡收发数据包情况

⑤nload

也是查看网卡流量，比sar更直观

⑥iostat

查看cpu和磁盘io情况

⑦iotop

查看那些进程占用io

1. linux的缓存与缓冲机制？

①cache（缓存）：是指读取出来的数据保存在内存中，当再次读取时，不用读取硬盘

②buffer（缓冲）：是指在写入数据时，先把分散的写入操作保存在内存中，当达到一定程度再集中写入磁盘，减少磁盘碎片和硬盘的反复寻道，加速数据的写入过程

清除缓冲：sync

清除缓冲：To free pagecache:

# echo 1 > /proc/sys/vm/drop\_caches

To free dentries and inodes:

# echo 2 > /proc/sys/vm/drop\_caches

To free pagecache, dentries and inodes:

# echo 3 > /proc/sys/vm/drop\_caches

缓存回收策略：

①基于空间，当缓存空间不足时

②基于容量，当缓存条数达到一定量时

③基于时间，一种是存活期，当存活一定时间自动清除，还有一种是空闲期，当有一段时间没有被访问过就清除。

缓存回收算法：

①FIFO：先进先出，用队列实现，在mysql面试题中有整理

②LRU：最近最少使用，用hash表+双向链表实现，更多用的是LRU-2而不是单纯的LRU在mysql面试题中有整理

③LFU：最不常使用，用队列+计数实现，在mysql面试题中有整理

关于redis和mysql的LRU算法实现在mysql面试题中整理了，下面来看看linux是如何应用什么策略清除缓存的——双LRU

即用两个LRU链表，一个activeLRU链表，一个inactiveLRU链表，数据页在两个链表之间可以移动，每次清除是清除inactive链表的尾部。

其实关于linux释放数据页也分回收和不回收（释放），回收又分回写磁盘和回收到swap分区等等知识，慢慢来。

1. 你说说swap分区是怎么回事吧？

定义：当物理内存不够用的时候，把硬盘空间的一部分释放出来以供当前运行的程序使用，意思就是有些很长时间没什么操作的程序把他放进swap分区（磁盘），进而达到增大内存的效果，如果被放进swap分区的程序再次运行，就恢复到内存中，比如电脑打开很多进程，突然回到很久没打开的程序中就会出现卡顿的现象！！

注意：并不是所有从物理内存中交换出来的数据都会被放到swap分区中，不然swap得死，有相当一部分数据被直接交换到文件系统或者直接释放，例如读文件时，因为是只读的，回收缓存时不存在脏数据，所以直接释放即可，而写文件时就需要回写到文件系统了（有文件背景的是“文件页”），然而那些new出来的对象、栈中的变量数据、代码段他们在文件系统中没有映射，所以是“匿名页”，这些匿名页的缓存回收时需要swap分区保存。

这就是刚才所说了linux在回收缓存的时候哪些回收哪些释放，回收的又分为回收到swap分区还是磁盘。

现在应该串起来了吧。

1. 你知道五种IO模型吗？

先了解几个概念：

①同步：调用一个功能，在该功能没有结束前，我死等结果

②异步：调用一个功能，不需要知道该结果，有结果后回调通知

③阻塞：就是调用函数，该函数没有接受完数据或者没有得到结果之前不会反回

④非阻塞，就是调用函数，该函数立即返回，通过select通知调用者

那么有人会问，同步和阻塞到底什么区别啊，这不看上去一样嘛？

注意：他们修饰的对象不同，同步与异步指的是调用函数或者功能这个调用的手段，而阻塞与非阻塞指的是被调用的那个函数

下面看一下linux下的五种IO模型：

①阻塞IO

进程会一直阻塞，直到数据拷贝完成，比如网络io，当数据通过网卡写入内存后系统会通知cpu执行中断程序cpu将数据从内核拷贝到用户进程空间，这个过程中就有recv()、recvfrom()、send等函数的调用，比如调用recv()之后会调用内核操作，等待数据和复制数据，这就是阻塞进程了，该线程会被挂起。

虽然这是我们一开始学网络编程时写的，但是满足不了生产者消费者的需要。如果每个连接都开辟一个通道，开销太大

②非阻塞IO

用户进程不断的主动询问内核数据是都准备好而不会发生阻塞挂起，如果没准备好就立即返回一个error信息，不断的询问直到数据准备好。

非常不推荐，非常占用cpu

③IO复用模型（事件驱动模型）

主要是依赖select和epoll，对于一个IO端口，两次调用、两次返回，比阻塞IO并没有什么优越性，关键是能够通过对多个IO端口进行监听！

这种方式，虽然也会阻塞进程，但是强就强在epoll可以同时管理多个socket连接并且将维护等待的socket队列和阻塞进程完全分离（也是比select高明的地方）进程调用完recv之后阻塞，而socket交给epoll来监听，当内核收到数据之后会通知socket，而epoll会通知进程进入就绪状态，获得cpu之后就执行，那么比阻塞IO高明之处在于不用进程自己管理socket连接，epoll来同时管理多个socket连接！！，不然你进城就得采用多线程、多进程的模式，刚才讲了，开销太大。

④信号驱动IO

⑤异步IO

简单来说就是数据拷贝的时候进程无需阻塞。可以干其他的。

注意：经过五种IO模型的介绍，现在应该对异步和非阻塞有了很好的理解了吧，异步就是我把一切托付给你，完成后通知我，但是非阻塞则是你来干但是我会监督你。同步就是不管阻塞非阻塞我就等着你完成。

1. select和epoll你了解吗？

上面既然说了IO多路复用，那么能说一下linux下的select、poll、epoll的联系和区别吗？

首先本质上select和poll没有大的区别，只不过poll没对针对单个进程打开最大连接数的限制，select在64位系统中限制是2048个连接。

要知道网络编程中io多路复用用的很多，比如nginx、redis都有epoll的支持。

首先我们先要理清请求和服务之间是怎么做的，也就是说数据进入服务器到真正被处理的过程：

①网卡收到网线传来的数据

②通过硬件的传输写入内存的某个地址上，此时操作系统（内核）才可以读取

③网卡想cpu发出一个中断信号，操作系统便能得知有新数据进来，中断信号是硬件发给cpu的，优先级很高，比如断电之后电容有一定电量，此时cpu会中断当前进程去内存中把数据回写到磁盘，为了保证数据不丢

④内核使用执行中断程度的cpu把内存中根据数据来判断这个数据是谁需要的将数据拷贝到用户进程，同时唤醒需要这个数据的阻塞的进程。

下面说一下为什么需要select和epoll？

比如编程时创建了一个socket对象，这个对象包含发送缓冲区、接收缓冲区、等待队列成员（各个进程），当执行到recv()时，操作系统把该进程移动到该socket的等待队列中，所以该进程就被阻塞了。

那么现在问题来了，一个recv()就是一个socket连接，但是要知道一台服务器要接受几千个连接呢，难道要用几千个线程？肯定不合适，开销不行，所以就需要一个东西可以管理多个socket连接！此时select和epoll就派上用场了。

select核心思想：

有一个socket列表，如果所有socket都没有数据，那么就挂起进程，一旦socket中有一个或者多个有数据进来了那么就唤醒进程，并且遍历socket，到底哪个socket有数据我除了那个。

实现：

比如现在又三个socket连接到了A进程，那么把A进程用引用分别加入到三个socket的等待队列中，当某个socket有数据，那么cpu唤醒进程A来处理，也就是把A进程从等待队列中移除方导工作队列中。

缺陷：

三次遍历，第一次：没有数据时要把进程A放入全部的sockets的等待队列中，第二次：当有数据时还要把A从所有sockets等待队列中移除，第三次：当A被唤醒后需要遍历socket才知道谁的数据来了。

epoll核心思想：

①功能分离：维护等待队列和阻塞进程分离。意义在于不必像select似的只要阻塞就添加到所有sockets等待队列。

②就绪列表：就绪列表rdlist保存进来数据的socket，就不必像select一样遍历所有sockets了

实现：

创建epoll对象，该对象该对象内部维护就绪列表，区别来了：epoll对象会把自己的引用放入所有sockets的等待队列而不是进程A，这样cpu执行中断程序的时候会唤醒epoll对象而不是操作进程A，同时epoll的就绪列表中有了有数据的socket引用，而关键点就在这里，进程A的阻塞和唤醒跟这个就绪列表rdlist有关，如果就绪列表为空就阻塞否则你懂的就处理呗，此时处理就不用遍历了因为只有有数据的socket才会在就绪列表中。

问个小问题，请问epoll用什么数据结构存储就绪列表和用什么数据结构管理epoll对socket的添加和删除。

①双向链表作为就绪列表的数据结构可以快速插入和删除

②那么监控socket用什么，快速插入和删除是必须的，还要便于搜索，防止重复插入，那么这个数据结构就是二叉搜索树，但是为了避免高度极端采用平衡二叉树，而红黑树就是一个缓冲，既是平衡二叉树又没平衡二叉树那么苛刻。

## network部分

1. TCP与UDP区别？

①TCP提供面向连接的、可靠的数据流传输，而UDP是非面向连接、不可靠的数据流传输

②TCP提供有序、无差错、不丢失、不重复的字节流传输；UDP传输单位是用户数据报

③TCP连接需要三次握手和四次断开，UDP不需要

④TCP提供拥塞控制和流量控制机制；UDP不提供拥塞控制和流量控制机制。

常见的基于TCP的协议有FTP、Telnet、SMTP、POP3、HTTP；基于UDP的协议有：DNS、SNMP、TFTP

1. OSI、TCP/IP、五层协议的体系结构以及各层协议？

OSI：7层：物理层、数据链路层、网络层、传输层、会话层、表示层、应用层

TCP/IP：4层：网络接口层、网际层、运输层、应用层

五层协议：5层：物理层、数据链路层、网络层、传输层、应用层

每一层包含的协议选择如下：

①物理层：RJ45、CLOCK、IEEE802.3（中继器、集线器）

②数据链路层：PPP、HDLC 、VLAN、MAC（网桥、交换机）

③网络层：IP、ICMP、APP、RARP、OSPF、IPX、RIP（路由器）

④传输层：TCP、UDP

⑤应用层：HTTP、FTP、DNS、Telnet、SMTP

每一层的作用如下：

①物理层：通过媒介传输bit，确定机械及点起规范（比特bit）

②数据链路层：将bit组装成帧和点到点的传递（帧frame）

③网络层：负责数据包从源到宿的传递和网际互联（包packet）

④传输层：提供端到端的可靠报文传递和错误恢复（段segment）

⑤会话层：建立、管理和终止会话（会话协议数据单元SPDU）

⑥表示层：对数据进行翻译、加密、压缩（表示协议数据单元PPDU）

⑦应用层：允许访问OSI环境的手段（应用协议数据单元APDU）

1. 那现在就有一个一直搞不明白的问题就是，交换机、路由器、网关的概念和用途？

①交换机

在计算机网络系统中，交换机是针对共享工作模式的弱点而推出的。交换机拥有一条高带宽的背部总线和内部交换矩阵。交换机的所有的端口都佳节在这条背部总线上，当控制电路收到数据包以后，处理端口会查找内存中的地址对照表以确定目的MAC（网卡的硬件地址）的NIC（网卡）挂接在哪个端口上，通过内部交换矩阵迅速将数据包传送到目的端口。目的MAC如不存在交换机才广播到所有的端口，接收端口回应后交换机会“学习”新的地址，并把它添加到内部地址表中。

交换机工作于OSI参考模型的第二层数据链路层。交换机内部的CPU会在每个端口成功连接时，通过ARP协议学习它的MAC地址，保存成一张ARP表。在今后的通讯中，发往该MAC地址的数据包仅送往其对应的端口而不是所有的端口，因此交换机可用于划分数据链路层广播，即冲突域；但它不能划分网络层广播，即广播域。

②路由器

路由器（router）是一种计算机网络设备，提供了路由与转送两种重要机制，可以决定数据包从来源端到目的端所经过的路由路径（host到host之间的传输路径），这个过程称为路由；将路由器输入端的数据包移送至适当的路由器输出端（在路由器内部进行），这成为转送。路由工作在OSI模型的第三层网络层，例如网际协议。

路由器的第一个作用是连通不同的网络，另一个作用是选择信息传送的线路。路由器与交换器的差别：路由器是属于OSI第三层产品，交换器是OSI第二程产品

③网关

网关gateway是连接两个网络的设备。

在传统的TCP/IP术语中，网络设备只分为两种，一种为网关，另一种是主机host。网关能在网络间传递数据包，但主机不能传送数据包。在主机中，数据包需经过TCP/IP四层协议处理，但是在网关只需要到达网际层决定路径之后就可以传送。

在现代网络术语中，网关gateway和路由器router的定义不同，网关能在不同协议间移动数据，而路由器是在不同网络间移动数据。对于以太网中的网关只能转发三层以上数据包，这一点和路由是一样的。而不同的网关中并没有路由表，它只能按照预先设定的不同网段来进行转发。网关最重要的一点是端口映射，子网内用户在外网看来只是外网IP地址对应着不同的端口，这样看来就会保护子网内的用户。

1. DNS域名系统请简述其工作原理？

当DNS客户机需要在程序中使用名称时，他会查询DNS服务器来解析该名称。客户机发送的每条查询信息包括三条：指定的DNS域名、指定的查询类型、DNS域名的指定类别。基于UDP服务，端口默认53，该应用一般不直接为用户使用，而是为其他应用服务，如HTTP、SMTP等在其中完成主机名到IP地址的转换。以www.baidu.com为例：

①客户机向其本地域名服务器发送DNS请求报文，问www.baidu.com的ip

②本地域名服务器收到请求后，查询本地缓存，如果有直接返回给客户端，假设没有该记录，则本地域名服务器以DNS客户的身份向根域名服务器发送解析请求www.baidu.com

③根域名服务器收到请求后，判断该域名所属域，将对应的顶级域名服务器.com的IP地址返回给本地域名服务器

④本地域名服务器向顶级域名服务器发送解析请求报文www.baidu.com

⑤顶级域名服务器收到请求后，将授权域名服务器baidu.com的ip地址返回给本地域名服务器

⑥本地域名服务器向授权域名服务器发起解析请求报文www.baidu.com

⑦授权域名服务器收到请求后，将www查询结果返回给本地域名服务器

⑧本地域名服务器将查询结果保存到本地缓存，同时返回给客户机

1. TCP的三次握手建立连接和四次挥手断开连接？

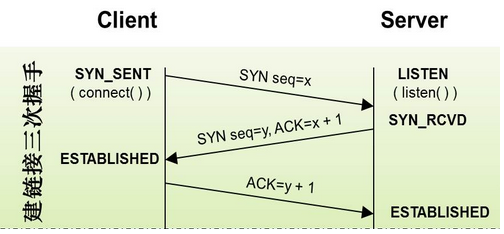
①TCP三次握手

1. 主机A发送位码为syn=1，随机产生seq number=x的数据包到服务器，客户端进入SYN\_SEND状态，等待服务器确认；主机B由SYN=1知道，A要求建立连接
2. 主机B收到请求后要确认联机信息，向A发送ack number（主机A的seq num+1），ack=1，随机产生seq num=y的数据包，此时服务器进行SYN\_RECV状态
3. 主机A收到后检查ack number是否正确（是否等于主机A的seq num+1）以及ack是否等于1，如果正确，主机A会再发送ack number（主机B的seq num+1），ack=1，主机B收到后确认seq num与ack=1则建立成功。客户端和服务器都进入established状态。

如果自己回答面试的话：

A（seq=x）——> B（ACK=1，ack=x+1, seq=y）——>A（ACK=1，ack=y+1），可能有一个疑问就是ack num和ACK，seq也是同样的问题。怎么有的发有的不发？

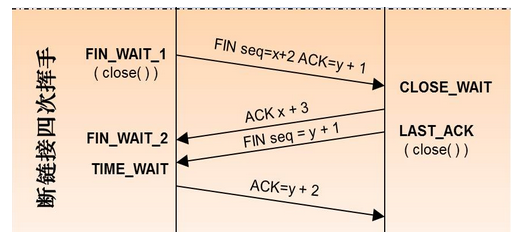
解答：我们首先明确一点就是ACK是确认信息，比如kafka的producer的acks参数，就是确认参数，所以我们认为携带ACK参数=1的都是响应的，所以第二次和第三次握手有ACK参数，而第一次没有；对于seq num序列号它是用来验证目标的，所以第三次握手没有意义了，已经确认完毕了就是他；同样的ack num=seq +1这个用来确定源的，所以第一次握手的时候还不知道源呢，从第二次服务器向客户端握手的时候才会让客户端来确认，所以ack num=seq+1参数在第二次和第三次握手才会有。



②TCP四次挥手

1. 主机A（可以是客户端也可以是服务器），设置seq num=x和ack num=y，向主机B发送一个FIN报文段；此时主机A进入FIN\_WAIT\_1状态，表示主机A没有数据要发送给主机B了
2. 主机B收到了主机A发送的FIN报文，向主机A回一个ACK报文段，ack num=seq+1；主机A进入FIN\_WAIT\_2状态，主机B告诉主机A，我也没有数据要发送了，可以进行关闭连接了
3. 主机B向主机A发送FIN报文段，请求关闭连接，同时主机B进入CLOSE\_WAIT状态
4. 主机A收到主机B发送的FIN报文段，向主机B发送ACK报文段，然后主机A进入Time\_WAIT状态；主机B收到主机A的ACK报文段之后就关闭连接，此时，主机A等待2MSL后依然没有收到回复，则证明Server端已正常关闭，那好，主机A也可以关闭连接了。

如果自己回答面试官的话：



主动请求关闭的一方的状态从FIN\_WAIT\_1 ——> FIN\_WAIT\_2 ——> TIME\_WAIT状态，被动关闭连接的一方的状态从CLOSE\_WAIT ——> LAST\_ACK。

A（FIN seq ack） ——> B （ack） B（FIN seq） ——> A（ack）

1. 流量控制和拥塞控制是什么以及怎么实现的？

①流量控制：数据的传送和接受过程中很可能出现接受方来不及接受的情况，这时就需要对发送方进行控制，以免数据丢失。流量控制用于防止在端口阻塞的情况下丢帧，这种方法是当发送或接收缓冲区开始移除时通过将阻塞信号发送回源地址实现的。流量控制可以有效的防止由于网络中瞬间的大量数据对网络带来的冲击，保证用户网络高效而稳定的运行。

实现：TCP采用大小可变的滑动窗口机制实现流量控制。窗口的大小是字节。在TCP报文段首部的窗口字段写入的数值就是当前给对方设置发送窗口的数据的上限。

②拥塞控制：网络拥塞现象是指到达通信子网中某一部分的分组数量过多，使得该部分网络来不及处理，以致引起这部分乃至整个网络性能下降的现象，严重时甚至会导致网络通信业务陷入停顿，即出现死锁现象。拥塞控制是处理网络拥塞现象的一种机制。

说白了就是大量数据涌入同一个交换节点（如路由器）导致该节点资源耗尽而必须丢弃后面到达的数据报，这就是拥塞。

实现：TCP同样采用滑动窗口机制对网络进行拥塞控制，将网络中的分组（TCP报文段作为其数据部分）数量维持在一定数量之下，当超过这个数值时，网络性能恶化。解决方案：慢开始、拥塞避免、快重传、快恢复四种算法。

注意：滑动窗口的意义不仅仅是解决这两个问题，窗口机制的存在可以让数据传输更加高效：一般理解，发送方发送报文，带有ack，接收方确认ack，发送方再继续发送，这样效率太低了，而使用窗口机制的话，发送方直接按照窗口大小预先划分好要发送的报文各个段，然后一起发送，然后等待接收方的反馈。

1. TCP重传机制？

思前想后，还是大篇幅的整理一下吧，一点一点来容易乱。

首先TCP在网络OSI的七层模型中的第四层：传输层，改成的数据叫做segment，IP在第三层网络层，改成的数据叫做packet，ARP在第二层数据链路层，改成数据叫做frame。

接下来看一下TCP头格式，20个字节。

第一个4字节：source port、destination port各16位

第二个4字节：sequence number32位

第三个4字节：ack number32位

第四个4字节：4位头长度、预留4位、8位标志位、16位window窗口大小

第五个4字节：TCP Flag状态机：16位检验和、16位紧急指针

关于什么是TCP的状态机，实际上就是标示当前tcp连接的状态！！我们在建立tcp连接和断开tcp连接的时候说过了主机A、B的各个状态了，这个地方就是标记连接状态的，到底是正在建立连接还是在传输还是正在断开连接等状态。

①建立连接时SYN（seqnum）超时

当server端收到SYN后返回SYN+ack后client掉线了，那么此时连接出于中间状态，linux下默认会重试5次，每次重试间隔时间翻倍，从1s开始，所以五次重试1 2 4 8 16，注意第五次发出后还要等32s才把连接断开，总共时间是1+2+4+8+16+32-1=63秒。

②关于SYN flood攻击

给server发了SYN就下线，于是server都会等63才断开连接，这样server的syn连接对垒耗尽了，于是linux给了一个tcp\_syncookies的参数来应对这件事，当syn队列满了，server会给每个连接发送一个cookie如果不响应就是攻击者。但是不建议使用这种方式，而是通过设置tcp\_synack\_retries减少重试次数、tcp\_max\_syn\_backlog增大syn连接数、tcp\_abort\_on\_overflow繁忙时直接拒绝连接。

③关于SYN的初始化

也就是sequence number的初始化，它是一个动态.首先为什么说是动态，如果每次seq都=1，现在seq已经到1000了，如果此时client断开然后又开始传seq=1，那么server可能就认为之前传的1000都是错误你要重新传。所以seq每4us加1，知道2^32次方，大概是4.5小时，所以只需要保证segment的最大存活时间MSL小于4.5小时即可。这样网络中就不会出现重复的seq。

④MSL和TIME\_WAIT

MSL max segment lifetime，segment的最大存活时间，在断开连接的时候主动发起方会有一个TIME\_WAIT的状态，从TIME\_WAIT到closed需要一个超时时间，而这个超时时间就是2\*MSL（30s），为什么要等30秒而不是直接closed呢？主要有两个原因：其一：TIME\_WAIT确保有足够的时间让对方接受到ACK，如果被动关闭的那方没有收到ACK，就会触发被动端重发FIN，一来一去正好是2\*MSL；其二：有足够的时间让这个连接不会跟后面的连接混在一起：有的路由器会缓存ip数据包，如果立即变成CLOSED状态，这个连接如果被重用了，那么就会发生缓存数据包发往新的连接。

⑤TIME\_WAIT连接太多

可以通过tcp\_tw\_resue和tcp\_tw\_recycle来缓解，但是最好不要打开。

TCP的重传：

要知道接收方恢复ack的时候是不能跳着回复的。所以如果要发送1 2 3 4 5共五分数据，接收方接收到1 2 后可能回复2 3，此时发送方直接发来了4，并没有预期的3，此时就要出发重传了。

①超时重传机制

一直不恢复ack给发送方，死等3数据，发送方一直等不到ack3，会重新发送3数据，然后接收方就可以恢复ack了。

弊端：浪费时间，收不到ack3，发送方可能就认为后发送的 4 和5 都丢了.....可能会重传3 4 5三分数据。

②快速重传机制

这种方式不像第一种是基于时间的！是以数据驱动的重传。也就是如果包没有连续到达，就ack最后那个可能被丢了的包，如果发送方连续收到3次相同的ack就重传。好处就是不用等timeout，因为不同的网络之间：局域网、互联网本身就有延时区别。

弊端：虽然解决了很大的时间问题，但仍然面临一个问题，数据3之后的4 5到底接收成功了还是失败了啊，到底是重传3还是重传3 4 5呢。

③SACK方法

selective acknowledgment，这种方式需要在TCP头里面加一个SACK的东西，用于记录丢失数据包后又收到了哪些数据包，比如上述的情况，接收方会反馈给发送方ack3+sack4-5，这样发送方就知道我只需要重传3就行。

④D-SACK方法——解决接收重复数据的问题

TCP的RTT算法：

上面我们知道TCP的重传依赖timeout很显然，timeout需要动态设置，TCP引入了RTT——round trip time，简单说就是发送数据包时记录t0，接收到ack记录t1，于是RTT=t1-t0。

这个RTT也有一个发展历程：

1. 最初的算法使用采样的方式计算平均RTT时间。但是有个问题就是到底是采用正常情况下的时间差还是采用重传时的时间差呢？
2. karn/partridge算法：不采用重传时的时间差，只采用正常情况时的时间差作为RTT。但是有一个巨大的问题就是当某一个时间段网络波动，那么由于该算法不采用重传时间差作为RTT，所以RTT不会因为此时网络波段而动态增大，所以这段时间所有的包都要重复的传！！这是个灾难！
3. jacobson/karels算法：将a的方式与当前最新时间差综合。

请介绍一下tcp的滑动窗口，刚才你说了那么多用处：

tcp头里面有4个字节的内容是给window用的，这个子弹用于告诉发送端子机还有多少缓冲区可以接收数据，于是发送daunt根据这个窗口值来处理数据，不会导致接收端处理不过来而丢包。

需要注意：可能存在zero window的情况，就是接收端不能再接收数据了，那么client怎么知道什么时候server可以继续接收啊，因为现在已经不发数据包了，TCP采用ZWPzero window probe技术，会发送ZWP包给接收方，持续3次，每次间隔30-60秒，如果这三次期间接收方都还不能继续处理，那么就断开连接。

注意：：：：DDos攻击：：只要有等待的地方就有DDos攻击，一些攻击者会在建立好http连接之后就把window设置为0，那么server就会等待并进行ZWP，那么如果大量这种情况就会耗尽server。

Silly window syndrome：糊涂窗口综合症，指的就是当server处理不过来时一直缩小window大小，那么client端就会做很多无用功，要知道TCP+ip头就有40Byte，MTU是1500字节，每次至少传输的数据有1460Byte，为了传输几个字节要那么大开销不值得。

我们说了TCP的slide window可以限流、可以提高数据发送和接受的并发量，还有一个作用就是解决拥塞问题。试想一下：如果出现了网络延时，tcp很有可能重传数据，迭代下去可能越来越恶劣，所以这种拥塞现象出现的时候TCP选择自我牺牲，自动降低自己的发送流量。

①慢启动算法

刚刚建立好的连接发送的数据量应该一点一点增加

②拥塞避免算法

避免了慢启动算法增长过快的弊端

③拥塞状态时算法

直接降低一半的发送数据量

④快速回复算法

主要用在发生快速重传的情况，要知道快速重传并不以为这网络延时（因为它不基于超时时间驱动），这种情况下不是降低一半流量而是将流量设置为阈值。

1. TCP连接的11中状态？
2. CLOSED：初始状态，表示关闭或者未打开
3. LISTEN：表示服务端的某个socket出于监听状态（比如调用了recv()）
4. SYN\_RCVD：表示收到了SYN请求报文，server端进入短暂的SYN\_RCVD状态，收到client返回的确认信息会进入ESTABLISHED状态
5. SYN\_SENT：表示发送了SYN请求报文的client端的状态，当接收到server端的sck和SYN报文后进入ESTABLISHED状态
6. ESTABLISHED：TCP连接已经建立
7. FIN\_WAIT\_1：
8. FIN\_WAIT\_2：这俩一起解释，这两个状态都是连接断开请求的发起方的状态（client和server都可），当发起方向另一方发起FIN报文后进入的FIN\_WAIT\_1状态，当对方回应ACK报文后进入FIN\_WAIT\_2状态，我们知tcp的四次挥手的第三次是由被动方发起的，如果被动方一直不发起第三次挥手，那么主动方会一直在FIN\_WAIT\_2状态直到系统重启
9. CLOSE\_WAIT：这是被动方的一个状态，这个状态非常重要！！！当主动方发起close断开连接的请求之后，被动方首先会立马回应一个ack，然后进入CLOSE\_WAIT状态，此时会检查是否还有数据需要发送，如果没有那么就进行第三次挥手，如果有的话就看程序了，可以继续发送数据
10. TIME\_WAIT：表示收到了对方的FIN报文，并且两方的数据都没有需要发送的了，可以进行关闭，那么发起方进入TIME\_WAIT状态
11. LAST\_ACK：当被动方接收到TIME\_WAIT发来的FIN报文之后就会进入LAST\_ACK状态，其实就是进入了CLOSED状态。也就是被动方先关闭。
12. CLOSING：这种状态很罕见，出现在两方都同时想关闭socket的情形下

最后：：关于TIME\_WAIT状态的超时时间2\*MSL，为什么这样设计，上面已经提到了，两个原因：第一让被动方足够时间收到ack，第二是避免连接重用时数据包的混乱。

1. 关于epoll和select之前介绍过了，你能总结以下epoll相对select的优势吗？你了解epoll的两种工作模式？

select劣势：

①最大连接数：64位系统最多2048个连接，因为

②效率：每次进行唤醒都会线性扫描所有的fd\_set(fd：文件描述符)

③内核和用户空间内存的拷贝

epoll优势：

①无最大连接数限制

②效率：通过将进程的挂起和socket的监听解耦达到每次处理的socket不需要轮询，epoll使用rdllist（双向链表）来告知哪些socket的数据就绪了，关于socket就绪队列是用红黑树存储的

③epoll使用共享内存页的方式：内核内存和用户空间内存共用内存的方式避免数据的多次拷贝，CopyOnWrite技术采用的也是共享内存页的方式

epoll两种工作模式：水平触发&边缘触发

①水平触发

内核中的socket接收缓冲区不为空，有数据刻度，读时间一直触发；内核中socket发送缓冲区不满，可以继续写入数据，写时间一直触发

②边缘触发

内核中socket接收缓冲区由空变为不为空，数据由不可读变为可读，仅触发一次；内核中socket发送缓冲区由满变为不满，数据由不可写变为可写，仅触发一次

区别和联系：

①LT水平触发很显然效率低，因为一直有epoll的系统调用通知你读和写，但是这种方式不会丢失数据

②ET边缘触发很显然减少了对epoll的系统调用，可以处理高并发的情况，但是由于是仅一次触发，所以存在数据读取不全、写入不完整（数据丢失）的情况，可以通过非阻塞读写的方式来处理（也就是死循环进行read和write来解决数据丢失问题）；另外对于多个socket的数据同时进入内核的情况epoll如何处理？由于epoll的ET模式的一次触发情况，多个socket数据进来可能只有一个socket连接被处理，那么其他连接可能不会被处理，所以程序中还应用有accept的外层循环，处理就绪队列中所有连接后再推出循环。

1. 大规模连接上来，并发模型怎么设计？

linux的并发模型有三种：多进程并发、多线程并发、IO复用模型

①多进程并发

accept返回成功时，就为了这个连接fork一个子进程

②多线程编发

...............................................................启用一个线程

关于多进程和多线程，多线程无疑会减少资源消耗和处理效率，但是设计并发安全问题时多线程就增加了复杂度甚至效率不及多进程。

③io多路复用模型

底层采用select/epoll的方式对socket进行监听和进程唤醒和挂起，不必针对每个连接都有独立的进程或者线程。典型的Nginx多进程+io复用模型。