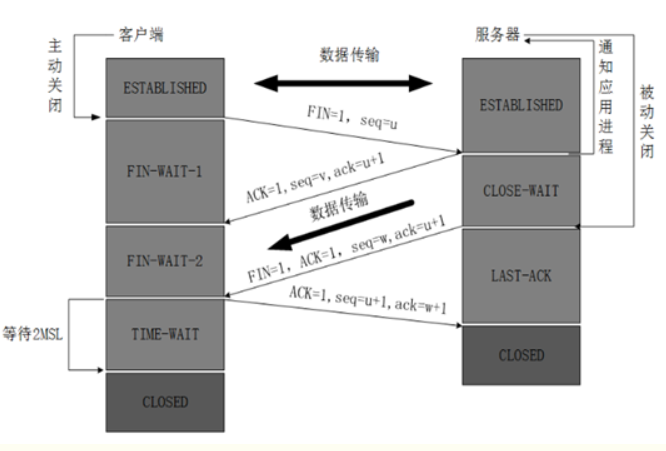
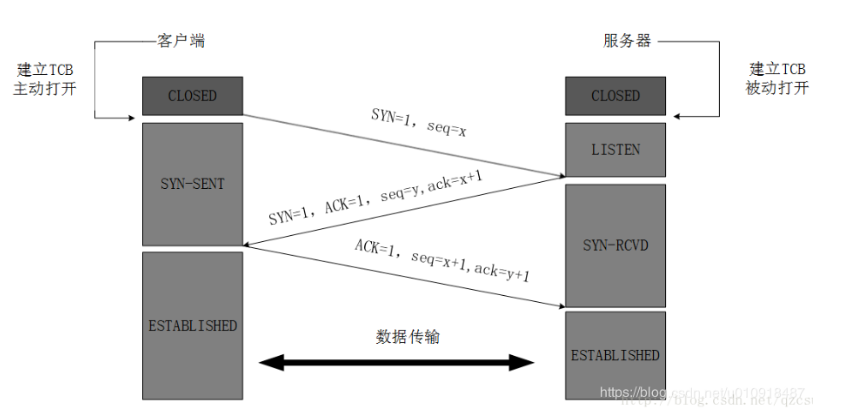
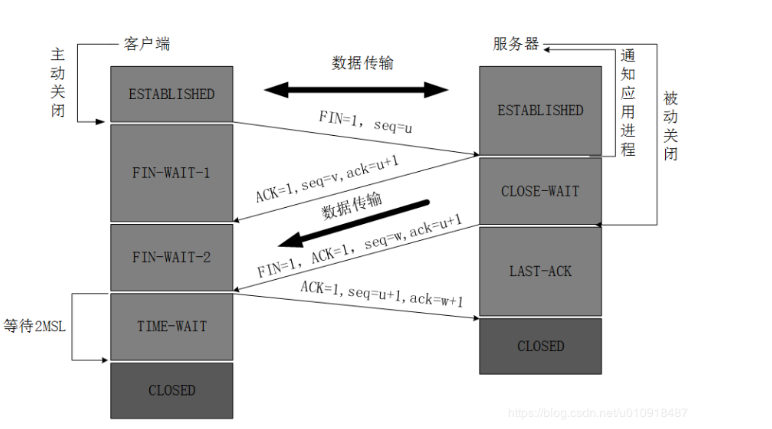
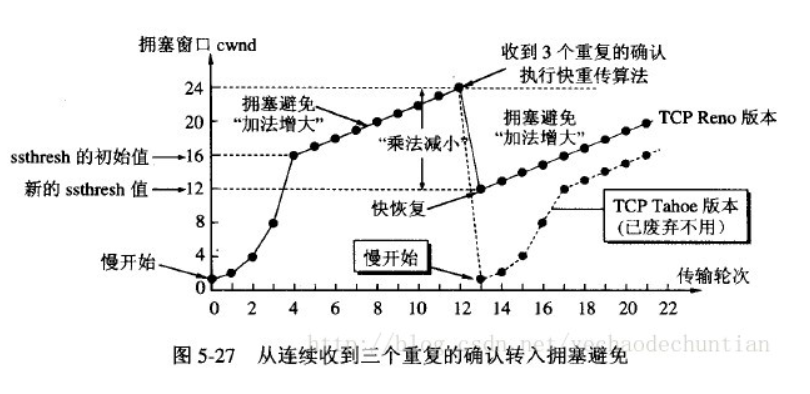
## 2020/08/03==================================

1. 自我介绍
2. 哈希表怎么实现，冲突怎么解决
   1. 哈希表的底层数据结构是数组，很多地方也叫Bucket。首先通过将key的值传给hash函数，求出对应的索引，找到相应的下标进行存储，时间复杂度是O（1）。
   2. 解决方法：
      1. 开放定址法
      2. 再hash法
      3. 链地址法（HashMap）
3. 维护一个堆（logn）
4. Nlogn
5. B树和B+树
   1. B树是多路平衡搜索树，它类似于普通平衡二叉树，区别是允许每个节点有多个子节点。B树为外部存储器（读写磁盘）设计，用于读写大块数据。
   2. 空间局部性原理：存储器的某个位置被访问，它附近的位置也被访问。
   3. B+树
      1. 叶子节点存储数据，非叶子节点并不存储数据
      2. 叶子节点增加了链指针
   4. 区别
      1. B树的非叶子节点保存key和value，而B+树的非叶子节点只保存key的副本，叶子节点保存value（data值）。B+树查询时间复杂的logn，B树则与位置有关。
      2. B+树叶子节点数据是用链表连起来的，可以做到区间访问性，访问磁盘某个位置，附件位置也被访问。
      3. B+树适合外部存储，key小，磁盘单次IO信息量大，IO次数少。
   5. Mysql的数据结构是B+树
6. 聚集索引和非聚集索引
   1. 聚集索引包含索引和数据，索引的叶子节点就是对应的数据。索引顺序和表中记录的物理顺序是一致的。
   2. 非聚集索引将索引和数据分开，索引的叶子节点指向数据的对应行，等于做了一个映射。索引顺序和物理顺序不一致。
   3. 一个表只能又一个聚集索引，通常设为主键。
7. MVCC多版本并发控制（Multi-Version Concurrency Control），实现对数据库的并发访问。MVCC是通过保存数据在某个时间点的快照来实现的，也就是同一时刻不同事物看到的相同表里的数据可能不同。从而实现并发控制。写写用锁，写读用mvcc。
8. DB日志
   1. 重做日志：持久性，记录事务执行后的状态。
   2. 回滚日志：原子性，保证事务发生前的版本
   3. 二进制日志：实现备份，是增量备份，只记录改变的数据。（备份）
   4. 错误日志：启动停止以及运行过程中的错误信息
   5. 慢查询日志：查询时间长或无索引的查询语句
   6. 通用查询日志：记录所有查询
   7. 中继日志：主从复制，读取主服务器的二进制日志，本地回放，实现同步。（复制）
9. 事务用在并发操作多张表时，保证数据的完整性。一方发生错误，回滚数据，保证两次操作的安全。特征：
   1. 原子性：同一个事务是一个不可分割的操作单元，要么全部成功，要么全部失败。重做日志。
   2. 一致性：事务操作的前后状态是一致的，符合逻辑运算。
   3. 隔离性：并发执行多个不同的事务之间互不干扰。
   4. 持久性：事务一旦提交，对数据库的改变是永久性的。回滚日志。
10. 事务的隔离级别
    1. 并发问题
       1. 脏读：一个事务读取了其他事务未提交的数据，读的是未提交。
       2. 不可重复读：事务两次读取数据修改，读的是已提交。
       3. 幻读：幻读是插入或删除操作，是已提交的。
    2. 隔离级别
       1. 未提交读：事务A写数据，事务B不可写但可读修改未提交的数据。
       2. 提交读：事务A写数据，禁止事务B读未提交的数据。
       3. 可重复读：事务A写数据禁止事务B任何操作，事务A读数据禁止事务B写事务，。
       4. 序列化：事务被定义为串行执行。
    3. 隔离级别对比
       1. 
    4. Mysql：可重复读；Oracle：提交读。
11. TCP和UDP
12. TIME\_WAIT
    1. 四次挥手，客户端和服务器都可以主动释放，以客户端为例：
       1. 客户端提出释放TCP请求，进入FIN\_WAIT\_1状态，向服务器发送FIN报文段。
       2. 服务的收到FIN报文，发送一个ACK报文，表时确认收到，此时处于半关闭状态，服务器进入CLOSE\_WAIT状态。客户端收到ACK报文进入FIN\_WAIT\_2状态。
       3. 服务器没有要发送数据时，发送FIN报文，由LAST\_ACK状态，转为LISTEN状态。
       4. 客户端收到FIN报文，向服务器端发送ACK报文，表时确认，客户端进入TIME\_WAIT状态，待2个最长报文寿命MSL后进入CLOSE状态。
       5. 图示：
    2. TIME\_WAIT状态作用：
       1. 主动关闭方发送的ACK包可能有延迟，从而触发被动关闭方重传FIN包，这样极端情况是2个MSL。
       2. 延迟发送的数据段会干扰新建立的连接，所以要等待。
13. URL、长度、字符、安全
14. Head请求，head方法和get方法相同，只不过服务器返回时不会返回方法体，用来检测超链接的有效性，和最近的修改。
15. HTTP
    1. Http和Https
       1. Http是超文本传输协议，Https增加了SSL（安全套接字层）协议用于加密传输。
       2. Http80端口，https443端口
    2. Https请求过程
       1. 客户端向服务端请求https连接；
       2. 服务端向客户端返回SSL证书，包含公钥；
       3. 客户端对证书进行验证，一般和本地的证书做比较，如果是信任的，客户端生成密钥，通过公钥加密发送给服务器；
       4. 服务器通过私钥解密得到对称加密的密钥
       5. 通过对称加密的密文通信。
    3. http1.0和http1.1
       1. 长连接：http1.1默认使用长连接，维持一个长连接，不需要每次建立TCP3次握手连接；
       2. 节约带宽：HTTP1.1支持只发送header信息，在收到继续响应后，在发送body；
       3. HOST域：web server上多个虚拟站点可以共享一个ip和端口。
    4. http1.1和2.0
       1. 多路复用：同一个连接并发处理多个请求；
       2. 二进制分帧：应用层和传输层之间，加入二进制分帧层；
       3. 首部压缩：对header数据进行压缩，网络传输更快；
       4. 服务器推送：客户端的一个请求，服务器可以发送多个响应。将客户端需要的资源一起推送，避免创建多次请求。
16. 进程、线程、协程
    1. 区别：
       1. 进程是程序运行和资源分配的基本单元
       2. 线程是CPU调度和分派的基本单元
       3. 协程是一个函数，可以暂停执行过程，类似于多线程调度。一个进程包含多个线程，一个线程包含多个协程，协程不是操作系统内核控制，是程序控制，所以不需要线程切换的资源消耗。
    2. 语言
       1. Go：函数前加上go关键字，这次调用就会在一个新的协程中并发执行；
       2. Python：通过yield/send实现协程。
17. Linux
    1. 查看进程、端口 ps显示进程 netstat显示端口
       1. ps –ef 显示所有进程
       2. ps –ef | grep 进程名 进程名查pid
       3. netstat –nap | grep 进程pid 进程pid占用端口
       4. netstat –nap | grep 端口号 端口查进程
    2. 杀死进程
       1. kill -9 进程号
       2. 默认状态是 kill -15（sigterm先释放资源，再停止，会被阻塞） -9（singkill）该信号不能被捕捉或忽略。
    3. 杀死进程原理
       1. 执行kill命令，默认就是kill -15，系统发送sigterm信号给程序，程序释放资源，然后停止。但是程序再做其他事情，比如正在处理IO的时候，不会立刻停止，sigterm信号阻塞。
       2. kill -9命令，系统发送sigkill信号给程序，强制杀死该进程。会留下不完整状态的文件。
    4. ps中 –A/-e显示所以进程 f显示程序间关系 grep查找
18. 方法的重载（overload）和重写（overwrite）
    1. 重载是一个类定义多个同名方法，他们的参数不同。
    2. 重写是子类继承父类，子类定义一个方法与父类有相同的名称和参数，子类对象使用这个方法，会调用子类中的定义。
    3. C++的多态：在基类的函数前加上virtual关键字，在派生类中重写该函数。
19. 使用iterator进行遍历，erase进行删除
    1. 对于vector，erase返回下一个iterator，用while去循环删除；
    2. 对于map，删除iterator只影响当前iterator，所以for就行了。
20. auto\_ptr,shared\_ptr,weak\_ptr,unique\_ptr
    1. 只能指针的作用是自动释放内存空间，避免内存泄漏
21. Redis
    1. Redis是基于内存的、高性能的非关系型数据库。
    2. 内存的速度比磁盘快很多，系统访问数据库，先访问内存的缓冲区查数据，如果缓冲区没有，在到磁盘数据库操作。
    3. Redis持久化
       1. Redis是基于内存的，一旦重启数据会丢失，所以需要进行持久化操作，RDB（Redis DataBase）、AOF（Append Only File）。
       2. RDB是基于快照的，把所有数据保存到RDB文件中，SAVE、BGSAVE、config配置文件3种方式实现，SAVE会阻塞，BGSAVE不会阻塞，创建子线程，由子线程创建RDB，缺点是若父线程修改，则会丢失数据。
       3. AOF是Redis服务器执行写命令时，会将写命令保存到AOF文件中。命令追加到aof缓冲区，确认缓冲区写入文件，一般是1s同步一次，丢失数据少。缺点时文件大，恢复慢。默认aof恢复。
    4. BIO、NIO、AIO
       1. 系统IO分为两个阶段：等待和操作
       2. BIO同步阻塞IO，数据的读取写入必须阻塞在一个线程内等待其完成；
       3. NIO同步非阻塞IO，多路复用机制，一个线程复制轮询，查看IO操作状态并进行操作；
       4. AIO异步非阻塞IO，无需轮询，IO操作状态改变时，系统会通知对应的线程来处理。
    5. Redis和java的hash区别
       1. 都是数组加链表，Java的hashMap对于长度大于8的链表，转为红黑树；
       2. Redis有rehash操作，对于大量数据，Redis的hash性能高；
       3. Redis的hash冲突时从头部插入，O(1)，HashMap在1.6时头部插入，1-8是尾部插入，所以是O(n)。
22. 单例模式

## 2020/08/07==================================

1. 自我介绍
2. 浅拷贝和深拷贝
   1. 浅拷贝指向已有内存
   2. 深拷贝指向新内存
3. 空间换时间，加大CUP的吞吐量，内存的IO以64位为单位进行。如果64位的数据，从第1位开始读，而不是从第0位，CUP要进行两次IO。
4. 是
5. 自旋锁和互斥锁
   1. 自旋锁：不释放cpu，别的线程会自旋，尝试获得自旋锁，超过次数，挂起；
   2. 互斥锁：释放cup，其他线程挂起，知道操作系统唤醒它。
   3. 阻塞不释放cpu，挂起释放cpu。
   4. 悲观锁和乐观锁：
      1. 悲观锁：先锁定，再操作；分为共享锁和排他锁，多个事务公用一把共享锁，只能读不能写，如写数据时，变为排它锁，其他事务不能读写。
      2. 乐观锁假设数据一般情况不会冲突，所以在数据提交更新时加锁检测。实现方式：数据库版本号，乐观锁控制的标准增加时间戳。
6. 服务端开发能力：
   1. 语言的能力，从语法到编程能力，还有编程思维的理解；
   2. 数据库，熟练数据库的交互；
   3. 基础知识，数据结构，算法，网络，操作系统
   4. Linux编程，服务器的使用，主流的tomcat、apache这些，的使用和拓展；
   5. 应用方面，开发工具的使用，编译器，git、svn版本控制，其他的一些插件
   6. 个人能力，交流，项目经验

## 2020/08/07==================================

1. 自我介绍。
2. JVM内存模型：
   1. 程序计数器：每条线程有独立的计数器，记录线程执行的位置；
   2. Java虚拟机栈：基本数据类型和对象的引用；
   3. 本地方法栈：调用的本地方法，也就是非Java语言的方法，和Java虚拟机栈类似；
   4. 堆：对象的实例；新生代（eden+2个survivor（from,to），8：1：1），老年代（15次gc）
   5. 方法去：常量和静态变量，class文件。
3. GC回收：
   1. Java堆中的新生代和老年代，新生代进程被gc回收，老年代较少。永久代在方法去，Jdk1.8取消了永久代，改为元数据，也是方法区。
   2. 回收机制：年轻代Eden区和2个survivor区，from和to；
      1. 新对象分配在Eden区；
      2. Eden区满了，存活的对象复制from区，放不下则全放老年代，Eden区内存回收；
      3. Eden区又满了，Eden和from存活的对象复制to区，放不到放老年代，Eden和from回收；
      4. 复制15次，放老年代，老年代满，调用Full GC（老年代满，永久代满，主动调用），前面时Minor GC。
4. 调用Miror GC和Full GC；
5. G1：Garbage First：把堆分为很多区域（Region），初始标记，并发标记，最终标记，筛选回收。
   1. 初始标记：标记GC Roots直接关联的对象，方法区、JVM栈、本地栈引用的对象；
   2. 并发标记：从GCRoots出发，找出存活对象，和用户线程并发执行；
   3. 最终标记：用户线程变动的对象；
   4. 筛选回收：对各个region区域回收价值和成本排序，指定回收计划。
6. 线程池：可以创建若干个线程放在池子中。
   1. 参数：
      1. corePoolSize：核心线程数
      2. queueCapacity：阻塞队列大小
      3. maxPoolSize：最大线程数
      4. keepAliveTime：线程空闲时间
      5. rejectedExecutionHandler：任务拒绝处理器
   2. 线程池执行过程：
      1. 当前任务<核心线程数，创建线程；
      2. 大于core线程，队列不满，任务放入队列
      3. 大于core线程，队列满，
         1. <max线程，创建线程
         2. >=max线程，抛异常RejectedExecutionException，拒绝任务。
   3. 四种池：
      1. newCachedThreadPool：可缓存线程池，灵活回收线程
      2. newFixedThreadPool：固定长度线程池，控制最大并发
      3. newScheduledThreadPool：定时线程池
      4. newSingleThreadExecutor：单线程线程池，唯一工作线程，指定顺序执行
7. 拒绝策略：
   1. AbortPolicy：丢弃任务并抛出异常RejectedEcexutionException
   2. DiscardPolicy：丢弃任务不抛出异常
   3. DiscardOrdestPolicy：丢弃队列最前面的任务，重新提交
   4. CallerRunsPolicy：由调用者处理该任务
8. hashMap
9. Spring相关
10. TCP和UDP区别
    1. 面向连接，可靠性，实时性，首部开销，点到点
11. TCP和UDP，假设客户端主动发起。
    1. TCP三次握手
       1. 客户端发送编号SYN1和随机序列号seq=J，SYN\_SEND状态；
       2. 服务端由SYN=1，就知道要建立请求，标志位SYN和确认值ACK = 1，确认值编号ack=seq+1=J+1，随机产生序列seq=K，发送给客户端，SYN\_RCVD状态；
       3. 客户端检查ack=J+1，ACK=1，正确则标准ACK=1，ack=k+1，发送给服务端，服务端检查ACK=1，ack=k+1，建立连接，ESTABLISHED。
       4. 3次
          1. 客户端发送请求，SYN=1，seq=x；
          2. 客户端发送请求，SYN=1，ACK=1，ack=x+1，seq=y；
          3. 客户端发送请求，ACK=1，ack=y+1。
       5. 
    2. TCP四次挥手
       1. 客户端发送断开请求FIN=1、seq=x，FIN\_WAIT状态；
       2. 服务端发送确认ACK=1，ack=x+1，seq=y；CLOSE\_WAIT状态；
       3. 服务器发送断开请求FIN=1，ACK=1，seq=z，ack=y+1，LAST\_ACK状态；
       4. 客户端确认断开，TIME\_WAIT状态，等待2个MSL，ACK=1，seq=z+1。
       5. 
12. TCP流量控制和拥塞控制
    1. 流量控制：滑动窗口大小，是端到端的，发送方发送太快，接收端无法接收，通过滑动窗口大小来控制接收窗口值，控制发送方发送数据的速率；
    2. 拥塞控制：解决过多的数据注入到网络，超负荷，是全局的；慢开始，拥塞避免、快重传、快恢复4个策略；
       1. 慢开始：窗口值设为1，每次增大一倍，触发门限限制则进入拥塞避免阶段；
       2. 拥塞避免：窗口每次加1，触发拥堵，窗口大小和门限限制变为一半；旧版本回到慢开始，新版到快恢复；
       3. 快重传：数据M2丢失，接收方重复发送M1的确认数据，发送发收到3次M1的确认以后，会立刻重发M2，同时触发快恢复；
       4. 快恢复每次加1。
       5. 
13. 协程
    1. 进程是