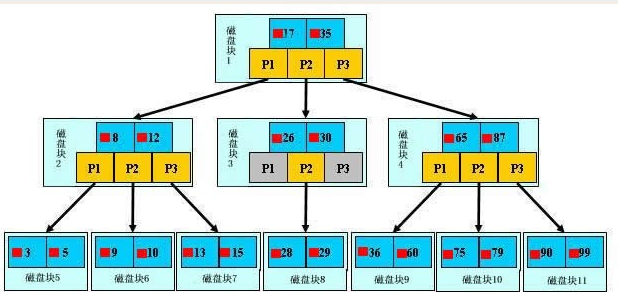
## mysql部分

1. Mysql索引原理？

索引的目的是提高查询效率，本质上是存储引擎用于快速定位记录的一种数据结构。

mysql采用的是B+数数据结构作为索引的存储。

介绍之前需要先知道磁盘IO和预读：磁盘IO是很昂贵的，局部预读性说明当计算机访问一个地址的数据的时候，与其相邻的数据也会被读到内存中。每次IO读取的单元是page一般是4k/8k，这就是一次IO，索引原理中一定要尽可能尽可能低的IO读取次数。



上图是B+数，浅蓝色代表数据块，可以认为是page页（不规范），深蓝色存储的是包含的数据项，黄色是指针，上图中磁盘块1包含数据项17和35，包含指针p1、p2、p3，其中p1表示小于17数据项的磁盘块，p2表示在17和35之间数据的数据块，p3表示大于35的磁盘块。真是数据全部储存于叶子结点3 5 9.....。非叶子节点不存储真是数据，只存储指引搜索方向的数据项，如17和35并不是真是的数据。

B+树查找流程：

如果要查找29，那么首先把磁盘块1加载到内存，此时发生了一次IO，在内存中用二分查找确定29在17-35之间，锁定p2指向的磁盘块3，将磁盘块3加载到内存中，此时发生第二次IO，同样的29在26和30之间，锁定磁盘块8，再发生第三次IO就把数据所有的数据找到了。

B+树性质：

①索引字段尽量要小，因为磁盘块大小一次IO最大是一个page，所以如果字段过大，树的高度越大，IO次数将越多

②索引的最左匹配，比如多个索引（name,age,sex），首先会根据name来决定下一步搜索方向，如果name缺失，将不知道走哪里，将会遍历所有记录了，所以如果有符合索引，查询的时候一个顶要按照顺序加where条件。

最左匹配有个特别之处就是当where中只有=和in的时候可以乱序，mysql自动识别来优化索引。

Mysql索引分类：

①普通索引index：加速查找

②唯一索引：

主键索引：primary key：加速查找+约束（唯一且不为空）

唯一索引：unique：加速查找+约束（唯一）

③联合索引：

-primary key(id, name)：联合主键索引

-unique(id, name)：联合唯一索引

-index(id, name)：联合普通索引

④全文索引full text：用于搜索很长一篇文章时效果好

⑤空间索引spatial：几乎不用

注意区分联合索引和索引合并的区别

索引的类型：

①hash索引

查询单条块，范围查询慢

②btree索引

创建索引原则：

①最左匹配原则，需要注意一点就是当遇到范围查询字段的时候索引匹配结束，所以where后面把范围条件放到最后去

②=和in可以乱序

③尽量选择区分度高的字段做索引count(distinct col1)/count(\*)

④所以列不能参与计算

索引无法命中的情况：

①like ‘%xx’ 左模糊

②索引使用函数

③or：如果or两边都是索引的话没事

④类型不一致，比如string字段查询时不加引号也会不走索引

⑤普通索引的!=和范围查询不会命中索引，特例：字段类型是int的时候会走索引，

⑥排序字段为索引时select 的字段也需要是索引字段

⑦组合索引

⑧组合索引最左前缀原则(name, age)的连联合索引，where name=..and agel=...会使用索引，where name = ...会使用，但是where age = ...不使用索引

最后再说一下覆盖索引：

在where条件中用到了索引并且命中了，如果此时select中的字段含有这个索引字段，那么这就叫做覆盖索引，会提升查询速度的！！

慢查询优化的基本步骤：

①先看看是不是真的慢，注意设置SQL\_NO\_CACHE

②where条件单标查，锁定最小返回记录表。意思就是把查询语句的where都应用到表中返回的记录数最小的表开始查起，单表每个字段分别查询，看哪个字段的区分度最高

③explain查看执行计划，看是都与②中预期的一样（从锁定记录最少的表开始查询）

④order by limit 形式的sql语句让排序的表优先查

⑤加索引参照上述的索引规则

1. Mysql锁机制？

锁分类：

①按照对数据操作的类型分

共享锁（读锁）：多个读操作互不影响

排它锁（写锁）：当前写操作没有完成之前会阻断其他的写锁和读锁（注意：这里到底是锁哪里呢，是整个表还是某个行要搞清，因为假如只是行锁的话是不影响其他数据的读和写的！）

②按照对数据操作的粒度分

表级锁

行级锁

页级锁

表级锁：偏向MyISAM存储引擎，开销小、加锁快、无死锁；锁定粒度大、发生冲突概率大，并发度低。

行级锁：偏向InnoDB存储引擎，开销大、加锁慢、会出现死锁；锁定粒度小，发生冲突概率低，并发度高。

页面所：开销和加锁时间介于表锁和行锁之间、会出现死锁；锁定粒度也是介于之间。并发度一般。

MyISAM与InnoDB比较：

MyISAM存储引擎由于自身不支持事务，所以用的少，对于锁也相对比较简单：

表级共享锁：不影响其他用户对数据的读取，但是会阻塞其他用户对数据的写！

表级排它锁：阻塞所有对该表的操作！

所以适合读。

InnoDB存储引擎比较复杂，下面进行详细介绍。

InnoDB行锁的实现方式？

InnoDB行锁是通过给索引上的索引项加锁来实现的，只有通过索引条件检索数据，InnoDB才是用行级锁，否则InnoDB将使用表级锁，在实际应用中，要特别主要InnoDB行锁这一特性，不然的话，可能导致大量的锁冲突，从而影响并发性能。

从这里可以看出意向锁存在的意义！

①在不通过索引条件查询的时候，InnoDB确实使用的是表锁，而不是行锁。

②由于InnoDB的行锁是针对索引加的锁，不是针对记录加的锁，所以虽然访问不同行的记录，但是如果是使用相同的索引建，会出现锁冲突。

③当表有多个索引的时候，不同的事务可以使用不同的索引锁定不同的行，另外，不论是使用主键索引唯一索引、普通索引，InnoDB都会使用行锁来对数据加锁。

④即便在条件中使用了索引字段，但是否使用索引来检索数据是有Mysql通过判断不同执行计划的代价来决定的，如果mysql任务全表扫描相率更高，比如对很小的表，它就不会使用索引，这种情况下InnoDB将使用表锁，而不是行锁。因此在分析锁冲突的时候，别忘了检查sql的执行计划，以确定是否真正使用了索引。

所以：这就是为什么查询语句最好要有where 索引条件！！！这样才会提高效率。

同时表的建立可以没有主键但是索引一定要建立！！！！

什么事间隙锁？

当我们用范围条件而不是相等条件检索数据，并请求共享或者排它锁是，InnoDB会给符合条件的已有数据记录的索引加锁；对于键值在条件范围但并不存在的记录，叫做“间隙GAP”，InnoDB也会对这个“间隙”加锁，这种锁机制就是所谓的间隙锁（Next-key锁）。

间隙锁虽然可以防止幻读，但是也会造成无辜锁定导致其他操作等待。

死锁问题？

当两个事务都需要对方释放资源时发生死锁，InnoDB自带检查机制，会将影响记录少的事务回滚，但是最好设置InnoDB\_lock\_wait\_timeout，这个值不仅适用于死锁，当大量查询并发等待挂起时可以减少压力。

如何避免死锁？

①在应用中，如果不同的程序会并发存储多个表，应尽量约定一相同的顺序来访问表，这样可以大大降低产生死锁的机会。

②在程序以批量方式处理数据的时候，如果实现对数据排序，保证每个线程按固定顺序来处理记录，也可以大大降低死锁的可能。

③在事务中，如果要更新记录，应该直接申请足够的级别锁，即排它锁而不是先申请共享锁，更新时再申请排它锁，因为当用户申请排他锁是，其他事务可能又已经获得相同记录的共享锁。从而造成锁冲突，甚至死锁。

④在repeatable-read隔离级别下，如果两个线程同时对相同条件记录用select for update加排它锁，在没有符合条件记录情况下，两个线程都会加锁成功，程序发现记录不存在就试图插入一条新的记录，如果两个线程都这么做就会出现死锁。这种情况下，将隔离级别改为read-commitred就可以避免

⑤当隔离级别为read-committed时，如果两个线程都先执行了select ... fro update,判断是否存在符合条件的记录，如果没有就插入记录，此时，只有一个线程会插入成功，另一个线程会出现锁等待，当一个线程提交之后，第二个线程会因主键重出错，但索然这个线程出错了却获得一个排它锁，这是如果有第三个线程又来申请排它锁，也会出现死锁。对于这种情况，可以直接做插入操作，然后再捕获主键重异常或者在遇到主键重错误时总是执行rollback释放获得的排它锁。

InnoDB也可以应用表级锁：

①当涉及表大部分数据都需要改变时

②当事务涉及多个表、比较复杂、可能引起死锁时

1. Mysql事务隔离你了解多少？

事务的四大特性（ACID）：

①原子性（Atomicity）：指事务包含的所有操作要么全部成功，要么全部失败回滚

②一致性（Consistency）：指事务必须使数据库从一个一致性状态变换到另一个一致性状态，也就是一个事务执行前后必须处于一致性状态

③隔离性（Isolation）：当多个用户并发访问数据库时，比如操作同一张表，数据库为每个用户开启事务，不能被其他事务所干扰，多个并发事务要相互隔离

④持久性（Durability）：指一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的，即便在数据库系统遇到故障的情况下也不会丢失提交事务的操作

不考虑事务性情况下会发生什么？

①脏读：指一个事务处理过程中读取了另一个未提交的事务中的数据

②不可重复读：指对于数据库中的某个数据，一个事务范围内多次查询结果不同，这是由于查询间隔被另一个事务修改了

③虚读（幻读）：幻读是事务非独立执行时发生的现象。比如事务A整体修改了一批数据，B在A后面新插入一条原始数据，A再次读取的时候发现：咦不对啊，这就是发生了幻觉

总结：脏读是读取了未提交的脏数据，不可重复读和幻读都是读取了另外事务已提交的数据，不同之处就是不可重复读指的是同一个数据项，而幻读针对一批数据体。

Mysql提供的四种隔离级别？

①Serializable：串行化，可以避免所有情形，说白了就是完全阻塞

②Repeatble read：可重复度，避免了脏读、不可重复读

③Read committed：读已提交，可避免脏读

④Read uncommitted：读未提交，任何情况都可能发生

1. 请列举MyISAM和InnoDB引擎的五大区别，越多越好？

①存储结构不同：

MyISAM在磁盘上存储三个文件：.frm(表定义) .MYD(数据) .MYI(索引)

InnoDB在磁盘上存储一个文件，如果比较大就是多个文件

②存储空间不同：

MyISAM可被压缩，存储空间小，支持三种不同的存储格式：静态表、动态表、压缩表

InnoDB需要更多的内存和存储，它会在主内存中建立其专用的缓冲池用于高速缓冲数据和索引

③可移植性、备份及恢复

MyISAM数据以文件的形式存储，所以在跨平台的数据转移中很方便，备份和恢复时可针对某个表进行

InnoDB免费的方案可以是拷贝数据文件、备份binlog或者用mysqldump

④事务

MyISAM强调性能，每次操作都是原子的，不提供事务支持

InnoDB提供事务，外部键等高级数据库功能

⑤AUTO\_INCREMENT

MyISAM自动增长必须是索引，如果是组合索引，自动增长列可以不是第一列

InnoDB自动增长同样需要是索引，如果是组合索引必须是第一列

⑥表锁差异

MyISAM只支持表索引，select update insert delete都是表锁，如果加锁后满足insert并发是可以在表尾部插入，不过要注意的是select加的是共享锁，不排斥其他共享锁

InnoDB支持事务和行级锁，行锁提高了并发能力，但是只有在命中索引的情况下才会进行行锁，否则将锁表

⑦全文索引

MyISAM支持FULLTEXT全文索引

InnoDB不支持全文索引，不过在5.6之后支持

⑧表主键

MyISAM允许没有任何索引的表存在

InnoDB如果没有主键和唯一索引就会自动生成一个6字节的主键，数据是主索引的一部分，附加索引保存的是住宿因的值

⑨表的行数

MyISAM保存有表的总行数

InnoDB没有

注意：如果是select \* from table，myisam效率很高，innodb会遍历表，但是如果加了where条件那么myisam和innodb一样

⑩外键

MyISAM不支持

InnoDB支持

1. MySQL的主从复制原理和流程？

主要是应用三个线程完成的：

①主binlog线程：记录所有改变数据库数据的语句放进master的binlog中

②从io线程：使用start slave之后，负责从master上拉取binlog内容放在自己relay log里

③从sql执行线程：执行relay log中的语句

1. InnoDB引擎的四大特性？

插入缓冲，二次写，自适应哈希索引，预读

①插入缓冲insert buffer：InnoDb使用insert buffer欺骗数据库，对于非唯一索引，并非实时的修改操作并非实时的更新索引的叶子页，而是把若干对统一页的更新缓存起来来做合并一次性更新操作，转化随机IO为顺序IO，提高写的性能。

②二次写Double write：为了解决partial page write的问题，当mysql将脏数据flush到data file时，先使用memcopy将脏数据复制到内存中double write buffer中，之后通过double write buffer再分两次，每次写入1MB到共享表空间，然后马上fsync到磁盘，避免缓冲问题。

③自适应哈希：mysql的heap存储引擎默认的索引是哈希索引，InnoDB存储引擎提出了另一种自适应哈希索引。咦，InnoDB不是B+树索引吗？是的，只不过InnoDB有自己的优化如果发现通过哈希索引能提高查询性能，那么它就会根据B+树缓冲建立自适应哈希，建立很快且不是将整个表都建立哈希索引。

④预读read-ahead：说白了就是说发现你总读某个page的数据，mysql就自动将下64个page或者这个范围内的剩余page全部预先读到内存。

1. MySQL中的varchar和char的区别以及varchar(50)中50的含义以及int(20)中的20？

varchar和char的区别：

char是一种固定长度的类型，varchar则是一种可变长度的类型。

char的长度在0-225之间，保存char类型时，右边用空格填充，检索时去掉尾部空格，索引效率高，可以用在一定确定的字段，比如手机号、身份证号等；而varchar，长度在0-65535之间，但是最大有效的长度由定义的长度和字符集确定，整体最大长度是65535个字节。保存varchar类型值时的长度是实际长度+1，这个1是记录实际长度用的，索引时保留尾部的空格，索引效率不高。

总结：varchar空间上合算，char时间上合算。

关于varchar(50)：

最多放50个字符，varchar(50)和varchar(200)存储”hello”所占空间一样，但是后者排序的时候消耗更多的内存。

int(20)的含义：

指显示字符的长度，跟varchar(20)不同，仍占4个字节，如果长度超过了20那么只显示20位，当然如果设置了未满填0的设置，那么未满20时前面填0，默认不填。

和varchar(20)还有一点不同就是int(10)和int(20)存储和计算都一样。

1. MySQL的三范式？

第一范式1NF：数据库表中的子弹都是单一属性的，不可再分。这个单一属性由基本类型构成，包括整形、实数、字符串、逻辑型、日期等。

第二范式2NF：数据库表中不存在非关键字段对任意候选关键字段的部分函数依赖、

第三范式3NF：在第二范式的基础上，数据表中如果不存在非关键字段对任一候选关键字段的传递函数依赖。

（说实话有点蒙）

1. MySQL到底应该知道哪些日志类型，他们有什么作用吗？

①事务日志之一redo log日志

事务日志主要用来保障数据库的ACID，尽可能降低宕机造成内存中数据丢失

事务日志会生成两个文件ib\_logfile0和ib\_logfile1，大小默认是5M，用于保存事务日志，在事务执行过程中数据的更改在未提交前都会先写到事务日志中，由于事务日志是连续的磁盘空间，IO性能高，可以保证数据及时写入事务日志。

事务日志包含redo log和undo log，基于InnoDB存储引擎的mysql之所以可以从崩溃中恢复正式依赖于事务日志，当数据库宕机重启后，重启mysql会自行检查事务日志，然后一次处理：

对于事务日志中未正常提交的事务，则会记录到undo log中，因为事务未正确执行完，必须回滚从而保证数据一致性；对于事务中已正常提交但未同步持久化到磁盘上则会记录到redolog中，mysql会重新执行一遍事务。

②事务日志之undo log：上面说了

③二进制日志binlog

是一个二进制文件，主要用于记录可能引起数据库数据更改的sql语句，会记录insert update delete grant等语句以及其执行时间、执行时长等额外信息，但是不会记录select show等语句，此时有一个面试题是关于log\_bin和sql\_log\_bin的问题？

log\_bin是只读的全局的配置项，而sql\_log\_bin是会话设置项，sql\_log\_bin只针对当前会话，具体有什么作用呢，比如在恢复数据的时候我们不想将恢复的大量操作记录到binlog中，为了提高性能而已，此时就可以应用sql\_log\_bin，设置为off，这样当前会话就不会记录binlog了。

④查询日志

会记录所有查询日志，不建议开启

⑤慢查询日志

跟查询日志相似只不过会记录查询时间超过long\_query\_time的那些查询语句。不建议开启，当然在调优的时候可以适当开启一段时间。

⑥中继日志relay log

中继日志主要用于主从复制中，在slace节点上开启用于从master同步二进制日志数据

⑦错误日志error log

记录mysql守护进程启动和关闭过程中产生的错误信息，运行中产生的错误信息

注意：mysql的日志都是采用的顺序IO的方式写，这样大大降低了因为写日志导致的性能下降。但是数据和索引信息是随机IO（insertbuffer可能缓解一下随机IO）。

1. 请解释下mysql中的float、double、decimal的区别？

其实从基本数据类型中可以知道float是单精度浮点数，double是双精度浮点数，单精度和双精度的区别在于有效位数，单8双16位。浮点的意思是不精准，存在四舍五入。

回到数据库，当我们建表定义字段类型的时候，他们三个都可以指定精度和标度，精度就是有效位数、标度就是小数点后面几位比如

CREATE TABLE test(f FLOAT(5,2) DEFAULT NULL,d DOUBLE(5,2) DEFAULT NULL,de DECIMAL(5,2) DEFAULT NULL);

INSERT INTO test(f,d,de) VALUES(1.234,1.234,1.234);没问题

结果是

如果不指定标度和精度：

INSERT INTO test(f,d,de) VALUES(1.234,1.234,1.234);

结果是

float和double没问题，但是decimal默认是有精度和标度的（10,0）所以默认decimal没有小数点。

注意：sum()的时候float和double由于是浮点数不精准导致计算结果有很多位小数。

1. CHAR\_LENGTH、LENGTH、BIT\_LENGTH是什么意思？

BIT\_LENGTH：返回字符串的位数

LENGTH：返回字符串的字节数

CHAR\_LENGTH：返回字符串的字符数

可想而知Latin字符串的字符数和字节数是一致的，因为Latin一个字符就是一个字节编码，而对于Unicode和其他编码他们就是不同的。

1. Mysql中的ENUM的用法是什么？

指的是字符串的枚举类型，可以认为是预定义的取值范围

比如create table tt(name ENUM(‘Sam’,’Loe’,’Mike’) default ‘Leo’);如果插入了不是这三个值中的值，那么记录时null值。

1. LIKE和REGEXP操作有什么区别？

REGEXP：正则匹配，子串匹配（不区分大小写，如果想区分需要加上BINARY）

LIKE：完全匹配

1. Mysql的group by的实现原理，如果可以再说说distinct？

group by实际上会进行排序操作，与order by相比，group by主要只是多了排序之后的分组操作。在Mysql中group by有三种实现方式，两种利用了索引，还有一种完全无法使用索引的方式。

①使用松散索引扫描（Loose）

松散索引扫描就是当MySQL完全利用索引扫描来实现group by的时候，并不需要扫描所有满足条件的索引键即可完成操作得出结果。

对于group by引用的多个字段，需满足于所建立索引的最左前缀索引，否则进行group by操作时，无法利用索引。在利用索引时，group by可根据索引，即可对数据分组，此时完全不用去访问表的数据值（索引健对应的数据）。这种实现方式就是利用松散索引。

②使用紧凑索引扫描（Tight）

当group by引用的字段无法构成所建索引的最左前缀索引时，也就是说group by不能利用索引时。如何where语句（如果有的话）弥补了这种差距，比如：group by引用的字段为（c2，c3），而索引为（c1，c2，c3）。此时如果where语句限定了c1=a（某一个值），那么此时mysql的执行过程为先根据where语句进行一次选择，对选出来的结果集，可以利用索引。这种方式，从整体上来说，group by并没有利用索引，但是从过程来说，在选出的结果中利用了索引，这种方式就是紧凑索引。这种方式，mysql的执行计划为using where,use index。而松散索引的执行计划为using index for group by。

③临时文件

如果mysql如论如何都不能利用索引时，此时mysql将读取所有的数据建立临时表，对文件进行排序，完成分组操作。

关于distinct的实现原理实际上和group by非常类似，只不过实在group by之后的每组中只去除一条记录而已，同样有三种实现方式，仅仅使用索引的松散索引方式、紧凑索引方式以及临时表的方式，一点区别就是group by的临时表方式要进程filesort，不然没法分组，而distinct不用filesort，因为它只需要一条。

1. mysql缓存是如何实现LRU缓存淘汰策略的？

我之前只知道redis缓存的LRU，其实现在想想，Mysql同样也有缓存啊。

所以在介绍mysql实现缓存的LRU之前先说一说mysql的缓存吧。

mysql的缓存机制：

说白了就是缓存sql文本及结果集。用kv形式保存在服务器内存中，如果运行完全相同sql则会利用缓存，不会再去解析、优化、执行了。如果这个表修改了那么跟这个表相关的所有缓存都不在生效，显然如果频繁更新的表查缓存不合适而且还会给服务器造成压力。

命中条件：

绝对相同！！！刚才说了缓存在KV的hash表中，通过查询sql、查询数据库、客户端协议等作为key，在判断命中之前不进行解析。对于sql语句，任何的不同包括空格、注释等都会导致缓存不命中！同时如果存在不确定数据如now() current\_date()等这些查询不会被缓存。

工作流程：

①服务器接收到sql，以sql和一些其他条件作为key查询缓存表

②如果找到缓存直接返回缓存

③如果没找到，则进行sql解析、优化、查询等

④执行完sql后，将结果集缓存到缓存表中

缓存失效：

只要涉及表的修改的操作都会让跟表相关的缓存失效，另外就是LRU了等会介绍

缓存的内存管理：

缓存会在内存中开辟一块大小为query\_cache\_size的内存维护缓存，其中有40k空间维护元数据：空间内存、数据表和查询结果映射、sql和查询结果映射，mysql将这个大内存分为小内存块（query\_cache\_min\_res\_unit），每个小块中存储自身的类型、大小和查询的结果集、还有前后没存块的指针。

缓存机制与MyISAM和InnoDB:

需要注意的一点是：MyISAM只缓存索引不缓存结果集，但是os系统级别的缓存就不算了，但是InnoDB索引和结果集都缓存！但是只要遇到加锁的事务，那么该事务中的查询将不会查缓存而是进行sql解析、优化、查询。同时也要注意InnoDB是行锁+MVCC来进行多版本并发的，当前事务ID一定要大于的事务才可以查缓存！其实跟之前说表改变，那么涉及该表的缓存全部失效是一样的。

那么有人会问什么事MVCC？这个问题等会整理。现在来说一下Mysql缓存的LRU吧：

再说Mysql的LRU缓存实现之前先简单说一下缓存过期策略的三种策略：

①FIFO：先进先出，非常好理解，用一个FIFO队列就可以实现，新加入的进入队列尾部，淘汰队首的数据

②LRU：least recently used最近最少使用，用链表实现：新数据插入俩表头部，每当命中缓存就将该数据移动到链表头部，当链表满的时候删除链表尾部数据

如果用hash表+双向链表来实现LRU将更加高效，比如linkedHashmap，查询是否命中显然hash快，插入删除数据显然链表更快，所以...

但是LRU有一个缺陷就是当出现批量插入时容易造成缓存污染，应该可以理解吧，所以更优的方案是LRU-K，其中K一般是2即LRU-2，意思就是访问2次再移动到链表头部

③LFU：least frequently used最近频繁使用：用队列+计数实现，新数据加入队列尾部，计数为1，当队列中数据被命中时引用计数+1，同时队列重新排序，当需要淘汰数据的时候把队列尾部数据删除

知道了淘汰算法之后，实际上不管是redis还是mysql实现缓存淘汰的方式绝没有这么简单，所以还是有差异的，那么mysql到底是怎么进行LRU淘汰缓存的呢？

mysql用一个链表来实现LRU，跟上面介绍的差不多，但是区别在于每次新缓存进来的时候并不是插到链表表头而是5/8处，为什么？因为这个链表被分为young区和old区，old区占整个链表的3/8，每次插入到old区的开始，这样就会把old区尾部挤掉，如果old区的缓存被命中，就把这个数据移动到young区的表头，这样就让young区很少使用的向old区移动了。

顺便说一下redis的LRU缓存淘汰实现吧：

redis既支持LRU也支持LFU，对于LRU的实现redis是近似的LRU目的是节省内存。

可以通过设置maxmemory来设置开启缓存淘汰，如果设置为0，表示不限制内存，那么就不会触发缓存淘汰。近似的LRU实现就是按照预先设定好的一些参数，比如哪些key参数淘汰，比如选择有过期时间的key参与淘汰、所有key参与淘汰等多种方案，同时还会根据一个sample参数进行淘汰，这个sample参数越大越接近真实的LRU，所以不要太大，默认是5。

1. 什么是InnoDB的MVCC？

MVCC(Multiversion Concurrency Control)，即多版本并发控制技术，它是的大部分支持行锁的事务引擎不在单独的使用行锁来进行数据库的并发控制，取而代之的是把数据库的行锁与行的多版本结合起来，只需要很小的开销就可以是先非锁定读（非阻塞读），从而大大提高数据库性能。

已经知道的是，加了读锁后不能再加写锁，加了写锁后不能加任何锁，表锁开销低并发弱，读的表锁可以再读而写的表锁读写都不行，行锁开销大，并发强存在死锁。

MVCC主要是为Repeatable-Read事务隔离级别做的。主要实现原理：

InnoDB存储的最基本单元row中包含了一些额外的存储信息：

①DATA\_TRX\_ID，6个字节标记了最新更新这条记录的transaction id，每个事务自动+1

②DATA\_ROLL\_PTR，7个字节指向当前记录项rollback segment的undo log记录，找之前版本的数据就是通过这个指针

③DB\_ROW\_ID，6个字节当由InnoDB自动产生聚集索引时，聚集索引包括这个值，否则聚集索引中不包括这个值，这个用于索引中

④DELETE BIT位用于表示该记录是否被删除，这里的不是真正的删除数据，而是标记出来的删除，真正意义的删除实在commit的时候

具体执行过程：

begin -> 用排它锁锁定该行 -> 记录redo log -> 记录undo log -> 修改当前行的值，写事务编号，回滚指针指向undo log中的修改当前行

而InnoDB实现MVCC主要依靠两个附加字段：创建（修改）版本号、删除版本号

①insert

InnoDB为每个新增行记录当前系统版本号作为ID

②delete

InnoDB为每个删除行的巨鹿当前系统版本号作为行的删除ID

③update

InnoDB复制一行，这个新行的版本号使用了系统版本号，同时把系统版本号作为删除行的版本，说白了就是insert+delete

④select（关键）

InnoDB会检查每行数据，确保符合两个条件：第一：InnoDB只查找版本早于当前事务版本号的数据行，确保读已提交；第二：行的删除操作的版本一定是未定义的或者大于当前事务版本号的，确保行没有被删除

这样就实现了非阻塞读操作。

注意：InnoDB的MVCC并不是真真正正的MVCC，因为InnoDB在进行行操作的时候是有排它锁的禁止其他事务操作该行！然而理想的MVCC是很难实现的，因为要保证ACID。

## redis部分

1. 什么是redis以及有什么主要的特点，有什么应用场景？

Redis(Remote Dictionary Server)是C语言开发，基于内存的key-value数据库。

①速度快——内存、C语言、单线程

②丰富的value存储结构：string、hash、list、set、zset、bitsmap

③客户端语言实现多

④持久化：AOF RDB mix

⑤主从复制

⑥高可用——sentinel哨兵

⑦分布式——cluster

应用：

①缓存（会话缓存、数据缓存）

②Pub/Sub

③分布式锁

④分布式队列

⑤排行榜

既然说了redis丰富的value存储结构，那么就说一下他们类型、特点、应用场景吧？

①string

set、get、decr、incr、mget

用于计数：微博数、粉丝数

②hash

hget、hset、hgetall

其实就是双层结构被，就是再次封装，所以你认为是包含于被包含的关系就可以用hash

③list

lpush、rpush、lpop、rpop、lrange

双向链表，消息队列、存储微博关注列表、粉丝列表等

④set

sadd、spop、smembers、sunion

实现用hashmap实现的，跟hash数据类型一样，其实跟list差不多，先进先出但是区别有两点，一个是去重，第二个是重置，比如先插入顺序 a b b a ，正常来说只保存a b 但是由于插入顺序，运行spop时出来的顺序是b a

⑤sroted set

zadd、zrange、zrem、zcard

相较set而言，多了一个根据score排序，在实时场景如直播中的排行，比如粉丝数排行、礼物排行

⑥bitsmap

位图，肯定不陌生，在处理海量数据不重复位图、判重的bloomfilter已经认识到了它的强大，redis用bitsmap最主要的应用就是海量不同用户的计数！比如今天登录过的用户，瞬间就可以计算出来

⑦stream

5.0引入的，等会专门介绍一下吧，尤其是跟kafka、rabbitmq的对比

1. redis持久化机制请详细说一下？

①快照snapshots

默认情况下使用快照方式进行持久化，写入二进制文件dump.rdb，可以自行配置持久化策略，多少秒进行多少操作及进行快照，或者手动调用save bgsave

原理：redis进行forks，子进程将数据写入新的rdb文件，替换老rdb文件，从实现上讲：子进程dump数据的时候采用copy on write的方式，如果此时有新的client操作请求，并不会改变需要dump的数据而是改变一个copy备份数据。

save 900 1 #在900秒(15分钟)之后，如果至少有1个key发生变化，则dump内存快照。

save 300 10 #在300秒(5分钟)之后，如果至少有10个key发生变化，则dump内存快照。

save 60 10000 #在60秒(1分钟)之后，如果至少有10000个key发生变化，则dump内存快照。

RDB方式虽然可能出现间歇期数据丢失问题，但是大量数据重载效率很高。

注意：手动调用save的方式并不是上述forks子进程的方式而是在主线程中操作，所以会阻塞服务！

②AOF

以同步日志的方式记录操作，以类似WAL的方式将每次操作日志（写、删除、更改）以追加的方式写入.aof文件。目的是解决RDB方式间歇期的数据丢失问题。

appendfsync always #每次有数据修改发生时都会写入AOF文件。

appendfsync everysec #每秒钟同步一次，该策略为AOF的缺省策略。

appendfsync no #写日志什么时候sync到文件中没法保证。

每秒同步虽然可能损失一秒数据但是效率是最高的

注意：为了方式.aof文件过大，redis有rewrite的功能，实现起来跟RDB的方式非常类似，比如redis收到rewrite的时候，redis forks子进程，根据内存中的数据，往临时的aof文件中写入重建数据库状态的操作命令，此时父进程并不影响client的请求，但是需要将rewrite期间发生的写删命令缓存起来待rewrite操作完成追加到后面，保证了数据的一致性，最后用临时文件替换老aof文件

③mix——两者结合

4.0版本引入，区别在于aof每次rewrite的时候并不是像上述类似RDB的方式，而是将.rdb文件的内存以操作命令的方式写到aof的临时文件中，然后继续向这个aof文件追加dump之后的操作命令，这样就缓解了aof恢复慢，rdb间歇期丢失数据的问题。

1. redis集群模式请详细介绍一下，顺便说一下主从模式那就更好了？

首先说一下为什么要引入或者说为什么要使用集群模式，难道单机模式或者读写分离不行？

说集群模式之前先介绍一下redis的主从模式：

需要注意的一点是，redis单一的主从模式（比如读写分离）他的目的是缓解master的压力，将读的请求分散到所有slaves上。

主从模式：星型模型、线性模型

星型模型：读写分离，数据同步高效，但是主节点切换需要用哨兵

线性模型：读写分离，数据需要一个一个同步低效，但是主节点变更只需要将下一个节点变成master即可

而集群模式与单一主从模式最大的区别就是，一个cluster可以有多个master-slave，每个master存储的数据是不同的，slave是master的备份，可以读但是不能写。

复制原理——全量复制：

不管是单一主从还是集群模式都会涉及从对主的数据同步，下面说一下复制原理：

当slave启动时候会向master发送sync命令，master收到sync命令就会主动进行rdb的dump，并且缓存dump期间的请求，然后全部发送给slave，然后master每接收一条新的写请求都会同步给slave。

复制原理——增量复制：

看这个名字就明白了，master维护一个历史缓存，slave保存一个offset，这样就可以通过增量复制offset之后的数据即可。

星型模型中的哨兵sentinel：

sentinel的作用：

①监控master和slave是否正常运行

②master出现故障时自动将slave转换为master

redis-sentinel sentinel.conf

使用集群模式的优点：

①多master时由于存储数据不同，节点失效时只会失效一部分数据

②集群模式容灾性更好。一个实例一方面由于使用内存过大可能面临灾难数据失效，另一方面可能面临持久化时间长，阻塞对外服务（单线程）

③一台服务器上开启多个实例，可以充分利用多核cpu和内存

缺点：

使用集群模式最好要配以slave，因为集群模式如果发现某个master不可用而又没有salve来替代，那么整个集群就失效了！

1. 为什么单线程的redis如此之快？那为什么还要有nginx多进程的方式或者memorycache多线程的方式？

①完全基于内存，操作时间复杂度是O（1）

②单线程避免了cpu上下文切换和竞争消耗，不用考虑锁的问题，不可能出现死锁

③使用多路IO复用模型

mysql、meory cache、oracle（windows版）——单进程多线程

oracle（linux版）、nginx——多进程

nginx不涉及进程间、线程间数据的同步，因为nginx处理的都是不相关的数据，所以用多进程

1. redis缓存淘汰策略？

在整理mysql缓存淘汰、linux缓存淘汰的时候已经介绍过redis的淘汰机制了，linux采用双链表的方式进行LRU淘汰策略，mysql采用单链表的方式，但是插入是从5/8处插入。而redis当时整理的是近似LRU的淘汰策略。

六中缓存淘汰策略：

①volatile-lru：从已设置过期时间的数据集中挑选最近最少使用的数据淘汰

②volatile-ttl：从已设置过去时间的数据集中挑选将要过期的数据淘汰

③volatile-random：从已设置过期时间的数据集中任意选择数据淘汰

④allkeys-lru：从数据集中挑选最近最少使用的数据淘汰

⑤allkeys-random：从数据集中任意选择数据淘汰

⑥no-encition（驱逐）：机制驱逐数据——永不回收

在mysql面试题中整理redis缓存淘汰时说了LRU是近似的LRU，什么意思？

意思就是当sample值越大约接近真实的LRU算法，比较小的时候LRU算法就接近ttl的淘汰策略，也就是按照过期时间排序，删除快过期的数据。因为真实的LRU是需要内存来保存链表等数据结构的，所以为了节省内存以及数据移动的性能消耗。

1. redis的hash槽你了解多少？
2. redis的事务你了解吗，它和watch如何实现乐观锁CAS，跟mysql中的事务有什么区别？
3. redis如何实现的分布式锁，和zk的分布式锁有什么不同？
4. 请说一下什么是缓存穿透、缓存击穿、缓存雪崩？应该怎么解决或者避免？
5. 刚才你提到redis单线程不存在死锁的情况，也就是不存在key的竞争问题，但是可能存在key的并发问题！所谓并发就是可能存在多个set操作虽然都能执行，但是可能存在执行顺序的问题，比如正常 1 2 3 4 5 这个顺序set key，加入顺序变了 1 3 4 5 2，此时结果就不对了，redis怎么解决这种情况？