数据库面试题

MySQL

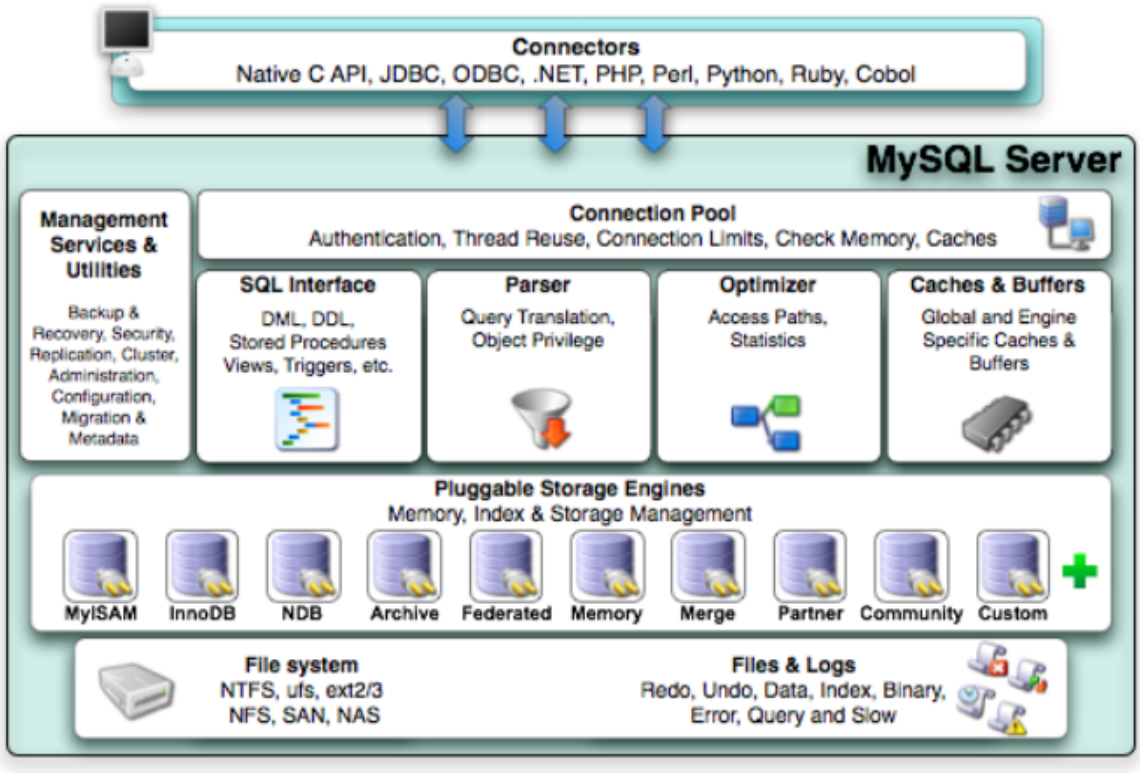
数据库面试题

MySQL

1.MySQL工作原理

1.1 mysql原理图各个组件说明：

- MySQL逻辑架构整体分为三层，最上层为客户端层，并非MySQL所独有，诸如：连接处理、授权认证、安全等功能均在这一层处理。
- MySQL大多数核心服务均在中间这一层，包括查询解析、分析、优化、缓存、内置函数(比如：时间、数学、加密等函数)。所有的跨存储引擎的功能也在这一层实现：存储过程、触发器、视图等。
- 最下层为存储引擎，其负责MySQL中的数据存储和提取。和Linux下的文件系统类似，每种存储引擎都有其优势和劣势。中间的服务层通过API与存储引擎通信，这些API接口屏蔽了不同存储引擎间的差异。



- 其八个组成部分及工作原理如下:

1. connectors
    - 与其他编程语言中的sql 语句进行交互，如php、java等。
2. Management Serveices & Utilities
    - 系统管理和控制工具
3. Connection Pool (连接池)
    - 管理缓冲用户连接，线程处理等需要缓存的需求
4. SQL Interface (SQL接口)
    - 接受用户的SQL命令，并且返回用户需要查询的结果。比如select from就是调用SQL Interface
5. Parser （解析器）
    - SQL命令传递到解析器的时候会被解析器验证和解析。
    - 主要功能：
        - 将SQL语句分解成数据结构，并将这个结构传递到后续步骤，后面SQL语句的传递和处理就是基于这个结构的
        - 如果在分解构成中遇到错误，那么就说明这个sql语句是不合理的，语句将不会继续执行下去
6. Optimizer (查询优化器)

- SQL语句在查询之前会使用查询优化器对查询进行优化(产生多种执行计划,最终数据库会选择最优化的方案去执行,尽快返会结果) 他使用的是"选取-投影-联接"策略进行查询。
- 用一个例子就可以理解
  - select uid,name from user where gender = 1;
  - 这个select 查询先根据where 语句进行选取，而不是先将表全部查询出来以后再进行gender过滤
  - 这个select查询先根据uid和name进行属性投影，而不是将属性全部取出以后再进行过滤
  - 将这两个查询条件联接起来生成最终查询结果.

7. Cache和Buffer (查询缓存)
   - 如果查询缓存有命中的查询结果，查询语句就可以直接去查询缓存中取数据。
   - 这个缓存机制是由一系列小缓存组成的。比如表缓存，记录缓存，key缓存，权限缓存等
8. Engine (存储引擎)
   - 存储引擎是MySql中具体的与文件打交道的子系统。也是Mysql最具有特色的一个地方。
   - Mysql的存储引擎是插件式的。它根据MySql AB公司提供的文件访问层的一个抽象接口来定制一种文件访问机制（这种访问机制就叫存储引擎）

1.2 SQL 语句执行过程

- 数据库通常不会被直接使用，而是由其他编程语言通过SQL语句调用mysql，由mysql处理并返回执行结果。那么Mysql接受到SQL语句后，又是如何处理
  - 首先程序的请求会通过mysql的connectors与其进行交互，请求到处后，会暂时存放在连接池（connection pool)中并由处理器（Management Serveices & Utilities）管理。
  - 当该请求从等待队列进入到处理队列，管理器会将该请求丢给SQL接口（SQL Interface）。
  - SQL接口接收到请求后，它会将请求进行hash处理并与缓存中的结果进行对比，如果完全匹配则通过缓存直接返回处理结果；否则，需要完整的走一趟流程：
    1. 由SQL接口丢给后面的解释器（Parser），解释器会判断SQL语句正确与否，若正确则将其转化为数据结构。
    2. 解释器处理完，便来到后面的优化器（Optimizer），它会产生多种执行计划,最终数据库会选择最优化的方案去执行,尽快返会结果。
    3. 确定最优执行计划后，SQL语句此时便可以交由存储引擎（Engine）处理，存储引擎将会到后端的存储设备中取得相应的数据，并原路返回给程序。

1.3 注意点

1. 如何缓存查询数据
   - 存储引擎处理完数据，并将其返回给程序的同时，它还会将一份数据保留在缓存中，以便更快速的处理下一次相同的请求。具体情况是，mysql会将查询的语句、执行结果等进行hash，并保留在cache中，等待下次查询。
2. buffer与cache的区别

- 从mysql原理图可以看到，缓存那里实际上有buffer和cache两个，那它们之间的区别：简单的说就是，buffer是写缓存，cache是读缓存。

3. 如何判断缓存中是否已缓存需要的数据
   - 这里可能有一个误区，觉得处理SQL语句的时候，为了判断是否已缓存查询结果，会将整个流程走一遍，取得执行结果后再与需要的进行对比，看看是否命中，并以此说，既然不管缓存中有没有缓存到查询内容，都要整个流程走一遍，那缓存的优势在哪?
   - 其实并不是这样，在第一次查询后，mysql便将查询语句以及查询结果进行hash处理并保留在缓存中，SQL查询到达之后，对其进行同样的hash处理后，将两个hash值进行对照，如果一样，则命中，从缓存中返回查询结果；否则，需要整个流程走一遍。

2.MySQL优化(简单)

- 数据库优化通常从三方面考虑:
    1. 替换有问题的硬件
        - 主要原因是数据库会占用大量资源,不过解决方案也仅限于此
    2. 对MySQL进程的设置进行优化
        - 适当分配内存,并让mysqld了解将会承受何种类型的负载
    3. 对查询进行优化
        - 对表应用了适当的索引.还可以记录慢查询
        - 优化MySQL查询缓存
        - 用EXPLAIN使你的SELECT查询更加清晰：了解MySQL正在进行什么样的查询操作，这可以帮助你发现瓶颈的所在，并显示出查询或表结构在哪里出了问题
        - EXPLAIN查询的结果，可以告诉你那些索引正在被引用，表是如何被扫描和排序的等等
        - 尽量避免SELECT *命令

3.MySQL读写分离

- 让master（主数据库）来响应事务性操作，让slave（从数据库）来响应select非事务性操作，然后再采用主从复制来把master上的事务性操作同步到slave数据库中。
- 多台MySQL服务器,分别提供读写服务,均衡流量,通过主从复制保持数据一致性.
- 由MySQL代理面向客户端,收到SQL写请求时,交给服务器A处理.收到SQL读请求时,交给服务器B处理.具体区分策略由服务设置.

4,MySQL主从同步

- MySQL主从复制是数据备份的一种手段,其用途主要有:实时灾备，用于故障切换;读写分离，提供查询服务;备份，避免影响业务.
- MySQL的主从复制方案是其自带的同步复制功能,并且它不是从硬盘上给文件直接同步,而是逻辑的binlog日志同步到本地的应用执行的过程。
- MySQL的主从复制是一个异步的复制过程（虽然一般情况下感觉是实时的），数据将从一个Mysql数据库（我们称之为Master）复制到另一个Mysql数据库（我们称之为Slave），在Master与Slave之间实现整个主从复制的过程是由**三个线程参与完成的。**其中有两个线程（SQL线程和IO线程）在Slave端，另一个线程（binlog dump线程）在Master端。

1.master: binlog dump线程

当主库中有数据更新时，那么主库就会根据按照设置的binlog格式，将此次更新的事件类型写入到主库的binlog文件中，此时主库会创建log dump线程通知slave有数据更新，当I/O线程请求日志内容时，会将此时的binlog名称和当前更新的位置同时传给slave的I/O线程。

2.slave: I/O线程

该线程会连接到master，向log dump线程请求一份指定binlog文件位置的副本，并将请求回来的binlog存到本地的relay log中，relay log和binlog日志一样也是记录了数据更新的事件，它也是按照递增后缀名的方式，产生多个relay log（host_name-relay-bin.000001）文件，slave会使用一个index文件（host_name-relay-bin.index）来追踪当前正在使用的relay log文件

3.slave: SQL线程

该线程检测到relay log有更新后，会读取并在本地做redo操作，将发生在主库的事件在本地重新执行一遍，来保证主从数据同步。此外，如果一个relay log文件中的全部事件都执行完毕，那么SQL线程会自动将该relay log 文件删除掉。

5.MySQL主动同步延迟问题

mysql的主从复制都是单线程的操作，主库对所有DDL和DML产生binlog，binlog是顺序写，所以效率很高，slave的I/O线程到主库取日志，效率也比较高，但是，slave的SQL线程将主库的DDL和DML操作在slave实施。DML和DDL的IO操作是随机的，不是顺序的，成本高很多，还可能存在slave上的其他查询产生lock争用的情况

由于SQL也是单线程的，所以一个DDL卡住了，需要执行很长一段事件，后续的DDL线程会等待这个DDL执行完毕之后才执行，这就导致了延时。当主库的TPS并发较高时，产生的DDL数量超过slave一个sql线程所能承受的范围，延时就产生了，除此之外，还有可能与slave的大型query语句产生了锁等待导致。

由于主从同步延迟是客观存在的，我们只能从我们自己的架构上进行设计， 尽量让主库的DDL快速执行。下面列出几种常见的解决方案：

1. 业务的持久化层的实现采用分库架构，mysql服务可平行扩展，分散压力;
2. 服务的基础架构在业务和mysql之间加入memcache或者Redis的cache层。降低mysql的读压力；
3. 使用比主库更好的硬件设备作为slave；
4. sync_binlog在slave端设置为0；
5. –logs-slave-updates 从服务器从主服务器接收到的更新不记入它的二进制日志;
6. 禁用slave的binlog。
7. 减少网络问题导致的延迟问题
    - –slave-net-timeout=seconds 单位为秒 默认设置为 3600秒
        - 参数含义：当slave从主数据库读取log数据失败后，等待多久重新建立连接并获取数据
    - –master-connect-retry=seconds 单位为秒 默认设置为 60秒
        - 参数含义：当重新建立主从连接时，如果连接建立失败，间隔多久后重试

6.MySQL三种主从同步区别

- 异步复制(Asynchronous replication)(默认)
    - 主库执行玩一次事务后,立即将结果返回给客户端,并不关系从库是否已经接受并处理.
- 全同步服务(Fully syschronous replication)
    - 当主库执行完一次事务, 且所有从库都执行了该事务后才返回给客户端
- 半同步复制(Semissynchronous replication)
    - 介于异步复制和完全同步复制之间
    - 主库在执行完一次事务后,等待至少一个从库收到并写到relay log中才返回给客户端

变种问题: 半同步复制与异步复制的区别, 在5.7做了哪些增强?

- 异步复制：主库执行完commit提交操作后，在主库写入binlog日志后即可成功返回客户端，无需等待binlog日志传送给从库。
- MySQL5.7之前的半同步复制：主库在每次事物成功提交时，并不及时反馈给前端应用用户，而是等待其中一个从库也接收到binlog事务并成功写入中继日志后（不等待从库应用这部分日志），主库才返回commit操作成功给客户端。

- MySQL5.7半同步复制增强：主库将事务写入binlog，传递到从库并写入从库的relay log，主库等待从库反馈接受到relay log的ack之后，再提交事务并且返回commit ok结果给客户端。

7.MHA高可用

MHA是一套优秀的实现MySQL高可用的解决方案.数据库的自动故障切换操作能做到在0~30秒之内,MHA能确保在故障切换过程中保证数据的一致性,以达到真正意义上的高可用.

- 主库挂了，但是主库的binlog都被全部从库接收，此时会选中应用binlog最全的一台从库作为新的主库，其他从主只需要重新指定一下主库即可(因为此时,所有从库都是一致的，所以只需要重新指定一下从库即可)。
- 主库挂了，所有的binlog都已经被从库接收了，但是，主库上有几条记录还没有sync到binlog中，所以从库也没有接收到这个event，如果此时做切换，会丢失这个event。此时，如果主库还可以通过ssh访问到，binlog文件可以查看，那么先copy该event到所有的从库上，最后再切换主库。如果使用半同步复制，可以极大的减少此类风险。
- 主库挂了，从库上有部分从库没有接收到所有的events，选择出最新的slave，从中拷贝其他从所缺少的events。

8.数据库事务

事务是最小的逻辑工作单元。

事务特性(ACID)

- 原子性:一个事务里面的操作要么不做,要么都做;
- 一致性:事务启动之前和启动之后要保持平衡状态
    - 例如完整性约束a+b=10，一个事务改变了a，那么b也应随之改变。
- 隔离性:在一个会话里面读取不到另一个会话里未提交的数据.
- 永久性:事务一经提交永不回退。

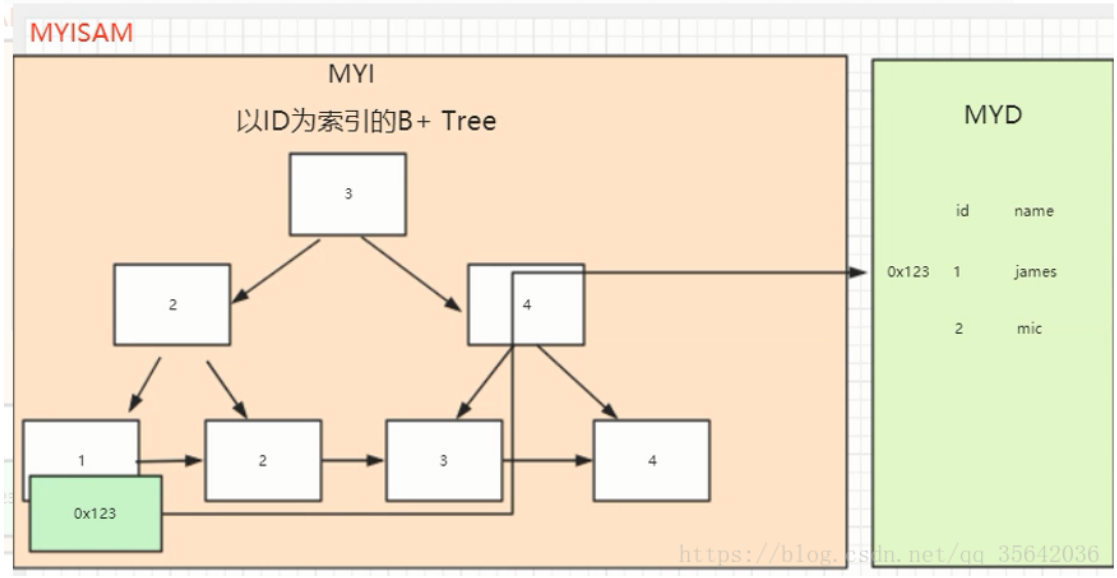
9.MySQL常用引擎与数据结构(Innodb与MyISAM)

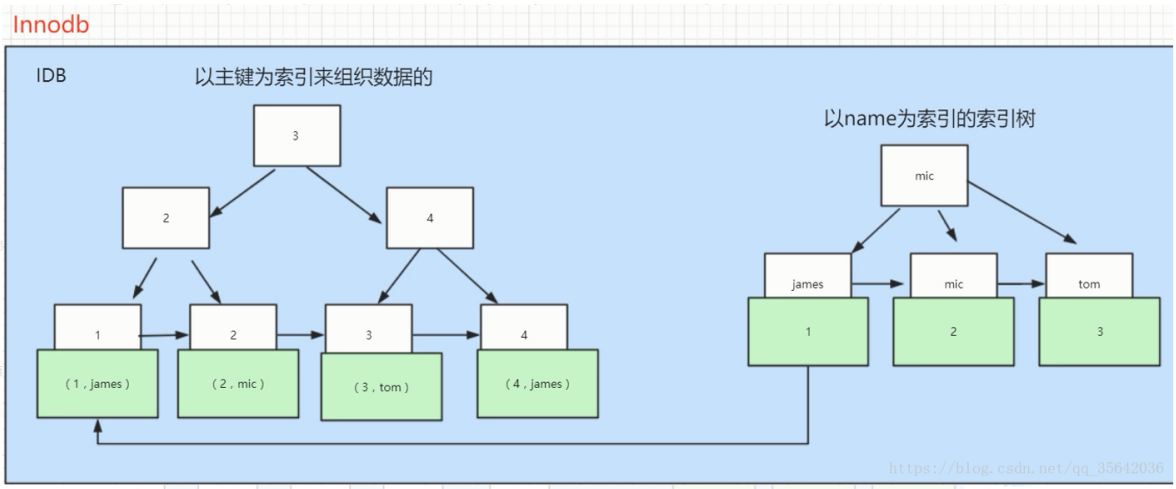
九点不同

1. InnoDB支持事务，MyISAM不支持
    - 对于InnoDB每一条SQL语言都默认封装成事务，自动提交，这样会影响速度，所以最好把多条SQL语言放在begin和commit之间，组成一个事务；
2. InnoDB支持外键，而MyISAM不支持
    - 对一个包含外键的InnoDB表转为MYISAM会失败；
3. InnoDB是聚集索引，使用B+Tree作为索引结构，数据文件是和（主键）索引绑在一起的（表数据文件本身就是按B+Tree组织的一个索引结构），必须要有主键，通过主键索引效率很高。但是辅助索引需要两次查询，先查询到主键，然后再通过主键查询到数据。因此，主键不应该过大，因为主键太大，其他索引也都会很大。

   MyISAM是非聚集索引，也是使用B+Tree作为索引结构，索引和数据文件是分离的，索引保存的是数据文件的指针。主键索引和辅助索引是独立的。

   也就是说：InnoDB的B+树主键索引的叶子节点就是数据文件，辅助索引的叶子节点是主键的值；而MyISAM的B+树主键索引和辅助索引的叶子节点都是数据文件的地址指针。





4. InnoDB不保存表的具体行数，执行select count(*) from table时需要全表扫描。而MyISAM用一个变量保存了整个表的行数，执行上述语句时只需要读出该变量即可，速度很快；
5. Innodb不支持全文索引，而MyISAM支持全文索引，查询效率上MyISAM要高；5.7以后的InnoDB支持全文索引了
6. MyISAM表格可以被压缩后进行查询操作
7. InnoDB支持表、行(默认)级锁，而MyISAM支持表级锁
    - InnoDB的行锁是实现在索引上的，而不是锁在物理行记录上。潜台词是，如果访问没有命中索引，也无法使用行锁，将要退化为表锁。

```
例如：
t_user(uid, uname, age, sex) innodb;
uid PK
无其他索引
update t_user set age=10 where uid=1;              #命中索引，行锁。
update t_user set age=10 where uid != 1;           #未命中索引，表锁。
update t_user set age=10 where name='chackca';     #无索引，表锁。
```

8. InnoDB表必须有主键（用户没有指定的话会自己找或生产一个主键），而Myisam可以没有
9. Innodb存储文件有frm、ibd，而Myisam是frm、MYD、MYI
    - Innodb：frm是表定义文件，ibd是数据文件
    - MyISAM：frm是表定义文件，myd是数据文件，myi是索引文件

如何选择

```
1. 是否要支持事务，如果要请选择innodb，如果不需要可以考虑MyISAM;
```

2. 如果表中绝大多数都只是读查询，可以考虑MyISAM，如果既有读也有写，请使用InnoDB。
3. 系统崩溃后，MyISAM恢复起来更困难，能否接受；
4. MySQL5.5版本开始Innodb已经成为Mysql的默认引擎(之前是MyISAM)，说明其优势是有目共睹的，如果你不知道用什么，那就用InnoDB，至少不会差。

其他

1. InnoDB为什么推荐使用自增ID作为主键?
    - 自增ID可以保证每次插入时B+索引是从右边扩展的，可以避免B+树和频繁合并和分裂（对比使用UUID）。如果使用字符串主键和随机主键，会使得数据随机插入，效率比较差。
2. innodb引擎的4大特性
    - 插入缓冲（insert buffer)
    - 二次写(double write)
    - 自适应哈希索引(ahi)
    - 预读(read ahead)

10.创建索引规则与注意事项

1. 避免索引过多，会影响写性能.
2. 给筛选效果低的字段加索引，几乎无效，如性别、状态标志等.
3. 每条查询执行时，只会使用一个索引，有需要时应该创建复合索引.
4. 复合索引使用时遵守“从左到右”原则，严禁左百分号.
5. 不要在索引字段上有运算操作和使用函数，将无法使用索引

11.SQL语句分类

1. DDL(Data Definition Language):数据定义语言，定义对数据库对象(库、表、列、索引)的操作。
    - 包括：CREATE、DROP、ALTER、RENAME、 TRUNCATE等
2. DML(Data Manipulation Language): 数据操作语言，定义对数据库记录的操作。
    - 包括：INSERT、DELETE、UPDATE、SELECT等
3. DCL(Data Control Language): 数据控制语言，定义对数据库、表、字段、用户的访问权限和安全级别。
    - 包括：GRANT、REVOKE等
4. Transaction Control:事务控制
    - 包括：COMMIT、ROLLBACK、SAVEPOINT等

12.Delete、truncate、drop都是删除语句，它们有什么分别?

1. delete 属于DML语句，删除数据，保留表结构，需要commit，可以回滚，如果数据量大，很慢。
2. truncate 属于DDL语句，删除所有数据，保留表结构，自动commit，不可以回滚，一次全部删除所有数据，速度相对较快。
3. Drop属于 DDL语句，删除数据和表结构,不需要commit，删除速度最快。

变种问题: 如果删除一个百万行的大表, 你如何删除?

13.MySQL原生支持的备份方式有哪些，并说出其优缺点?

1. 直接拷贝数据文件，必须是MyISAM表，且使用flush tables with read lock;语句，优点是简单方便，缺点是须要锁写，且只能在同版本的MySQL上恢复使用。
2. mysqldump，导出的是SQL语句，所以可以跨版本恢复，但是需要导入数据和重建索引，恢复用时会较长，如果是MyISAM表，同样需要锁表，如果是InnoDB表，可以使用--single-transaction参数避免此。

14.什么情况下应不建或少建索引

- 表记录太少
    - 如果一个表只有5条记录，采用索引去访问记录的话，那首先需访问索引表，再通过索引表访问数据表，一般索引表与数据表不在同一个数据块，这种情况下DB至少要往返读取数据块两次。而不用索引的情况下DB会将所有的数据一次读出，处理速度显然会比用索引快。
- 经常插入、删除、修改的表 对一些经常处理的业务表应在查询允许的情况下尽量减少索引。
- 数据重复且分布平均的表字段
    - 假如一个表有10万行记录，有一个字段A只有T和F两种值，且每个值的分布概率大约为50%，那么对这种表A字段建索引一般不会提高数据库的查询速度。

15.MySQL和Oracle的区别

1. Oracle是大型数据库而Mysql是中小型数据库，市场占有率达 同时Mysql是开源的而Oracle价格非常高。
2. Oracle支持大并发，大访问量
3. 安装所用的空间差别也是很大的，Mysql安装完后才152M而Oracle有3G左右，且使用的时候Oracle占用特别大的内存空间和其他机器性能。
4. Oracle也Mysql操作上的一些区别

   4.1 自动增长类型的处理
    - MYSQL有自动增长的数据类型，插入记录时不用操作此字段，会自动获得数据值。ORACLE没有自动增长的数据类型，需要建立一个自动增长的序列号，插入记录时要把序列号的下一个值赋于此字段。

   4.2 单引号的处理
    - MYSQL里可以用双引号包起字符串，ORACLE里只可以用单引号包起字符串。

   4.3 分页的SQL语句的处理
    - MYSQL处理分页的SQL语句比较简单，用LIMIT 开始位置, 记录个数；ORACLE处理分页的SQL语句就比较繁琐了。每个结果集只有一个ROWNUM字段标明它的位置, 并且只能用ROWNUM<100, 不能用ROWNUM>80

   4.4 长字符串的处理
    - ORACLE也有它特殊的地方。INSERT和UPDATE时最大可操作的字符串长度小于等于4000个单字节, 如果要插入更长的字符串, 请考虑字段用CLOB类型，方法借用ORACLE里自带的DBMS_LOB程序包。插入修改记录前一定要做进行非空和长度判断，不能为空的字段值和超出长度字段值都应该提出警告,返回上次操作。

   4.5 空字符的处理

- MYSQL的非空字段允许为空字符串，ORACLE里定义了非空字段就不容许为空字符串。

4.6 字符串的模糊比较

- MYSQL里用 字段名 like '%字符串%',ORACLE里也可以用 字段名 like '%字符串%' 但这种方法不能使用索引, 速度不快。

4.7 日期字段的处理

- MYSQL日期字段分DATE和TIME两种，ORACLE日期字段只有DATE，包含年月日时分秒信息，用当前数据库的系统时间为SYSDATE, 精确到秒

16.数据库的锁: 行锁, 表锁; 乐观锁, 悲观锁

- 表级锁：开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突的概率最高，并发度最低。
- 行级锁：开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低，并发度也最高
- 悲观锁：假定会发生并发冲突，屏蔽一切可能违反数据完整性的操作。
- 乐观锁：假设不会发生并发冲突，只在提交操作时检查是否违反数据完整性。不能解决脏读问题

17.MySQL服务器CPU跑满100%的情况分析

情况分析

1. 第一步,万能的重启.当然可能重启了一会儿还是继续高上去
2. 检查连接数和慢查询语句.开始分析了.
    - 我们的原则是,重启能解决的,绝对不开client
    - cpu100%通常情况下就是有慢sql造成的，这里的慢sql包括全表扫描，扫描数据量过大，内存排序，磁盘排序，锁争用等待等...
    - 一般表现现象sql执行状态为：Sending data，Copying to tmp table，Copying to tmp table on disk，Sorting result，locked
    - 通过show processlist查看当前正在执行的sql，当执行完show processlist后出现大量的语句，通常其状态出现Sending data，Copying to tmp table，Copying to tmp table on disk，Sorting result, Using filesort都是sql有性能问题
    - 可以用explain查看sql执行效率,分析索引

解决方案整理

1. Sending data表示：sql正在从表中查询数据，如果查询条件没有适当的索引，则会导致sql执行时间过长；
2. Copying to tmp table on disk：出现这种状态，通常情况下是由于临时结果集太大，超过了数据库规定的临时内存大小，需要拷贝临时结果集到磁盘上，这个时候需要用户对sql进行优化；
3. Sorting result, Using filesort：出现这种状态，表示sql正在执行排序操作，排序操作都会引起较多的cpu消耗，通常的优化方法会添加适当的索引来消除排序，或者缩小排序的结果集；
4. 出现sending data的情况：这种一般就是SQL不规范,优化SQL吧.
5. 检查网站是不是有被攻击之类的

变种问题1: 数据库高负载的排查和解决办法

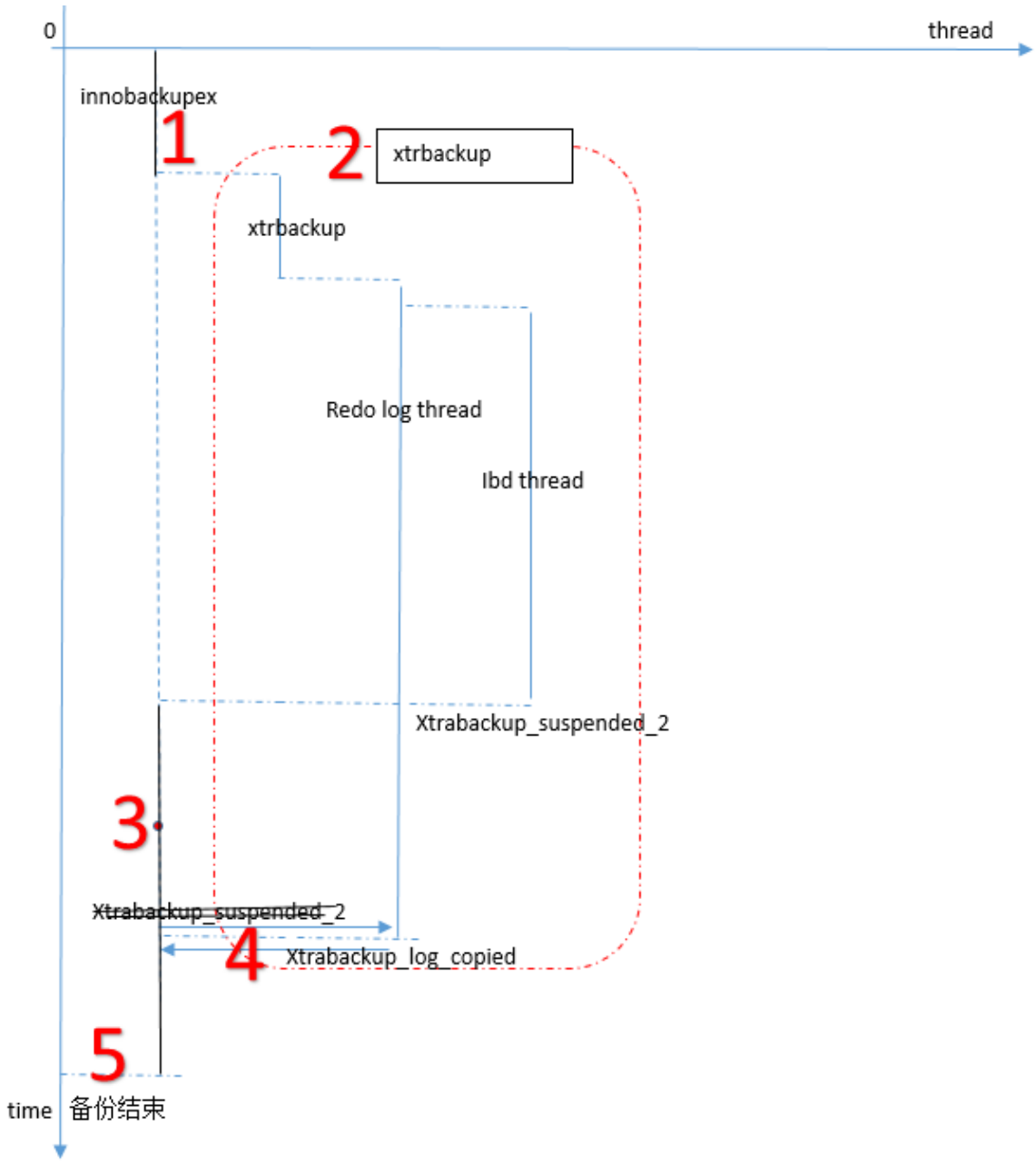
- 检查操作系统
    - 查看整体负载，使用命令w或者sar -q 1
    - 判断瓶颈在哪个子系统，使用命令w

    - 判断磁盘IO是否较大，使用命令：sar -d 1
    - 判断具体哪个进程消耗的磁盘IO最多，使用命令：iotop
- 检查MySQL层
    - 查看当前的MySQL查询语句，使用命令：mysqladmin pr|grep -v Sleep
    - 分析slow log
    - 分析慢查询语句

变种问题2:Explain执行计划中要关注哪些要素

1. type:本次查询表联接类型，从这里可以看到本次查询大概的效率
2. key:最终选择的索引，如果没有索引的话，本次查询效率通常很差
3. key_len:本次查询用于结果过滤的索引实际长度
4. rows:预计需要扫描的记录数，预计需要扫描的记录数越小越好
5. extra:额外附加信息，主要确认是否出现 Using filesort、Using temporary 类似情况

18.Innobackupex备份过程



innobackupex命令构成

1. Innobackupex内部封装了xtrabackup和mysqldump命令；
2. Xtrabackup是用来备份innoDB表的，内部实现对innoDB的热备份；
3. Mysqldump负责完成非innoDB表的备份；

innobackup备份过程说明

1. Innobackupex启动之后，首先启动xtrabackup进程，innobackupex进入等待状态；
2. Xtrabackup启动两个线程：redo log复制线程和ibd复制线程，redo log线程从最近的检查点开始复制，ibd线程开始复制innodb表的ibd文件。Ibd文件复制完成后，通过创建xtrabackup_suspended_2文件，唤醒innobackupex进程开始工作。Redo log线程继续工作。
3. Innobackupex进程检测到xtrabackup_suspended_2文件存在后，执行备份锁，取的一致性位点，开始复制非innoDB文件。非innoDB文件复制完后，innobackupex开始执行LOCK BINLOG FOR BACKUP，获取binlog的位置，并记录到xtrabackup_binlog_info文件中。
4. Innobackupex进程完成上述操作后，通过删除xtrabackup_suspended_2文件的方式通知xtrabackup进程，xtrabackup收到通知后停止redo log 复制线程，通过生成xtrabackup_log_copied文件的方式通知innobackupex进程。
5. Innobackupex进程检测到xtrabackup_log_copied文件存在后，释放锁资源，进行善后工作，结束备份。

总结：在没有锁等待等外部因素影响下，物理备份，就是对物理文件的copy过程，备份的速度和物理文件的大小和并发数和磁盘IO有关。而innobackupex命令只是在文件copy的过程中，增加了记录日志和一致性复制等功能。

19.BLOB和TEXT之前的区别是什么

- 在BLOB排序和比较中，对BLOB值区分大小写。
- 在TEXT文本类型中，不区分大小写进行排序和比较

1. BLOB是什么?
    - BLOB表示二进制大对象。
    - 可以保存可变数量的数据。
    - 根据所能容纳的值的最大长度，有四种BLOB类型:
        - TINYBLOB
        - BLOB
        - MEDIUMBLOB
        - LONGBLOB
2. TEXT数据类型是什么?
    - TEXT是不区分大小写的BLOB。四种文本类型是:
        - TINYTEXT
        - TEXT
        - MEDIUMTEXT
        - LONGTEXT

20.MySQL常见监控项(简单)

1.操作系统层面

- CPU

- 内存
- IO
- 网卡

2.MySQL层面

- MySQL是否运行
- 连接数
- qps
- tps
- 慢查询数
- 打开表数
- 当前脏页数
- 锁情况
- 主从同步状态（包括Slave_IO_Running、Slave_SQL_Running、Seconds_Behind_Master等

21.MySQL5.6, 5.7, 8.0版本各有什么特点

MySQL 5.6

1. 支持GTID复制
2. 支持无损复制
3. 支持延迟复制
4. 支持基于库级别的并行复制
5. mysqlbinlog命令支持远程备份binlog
6. 对TIME, DATETIME和TIMESTAMP进行了重构，可支持小数秒。DATETIME的空间需求也从之前的8个字节减少到5个字节
7. 支持Online DDL。ALTER操作不再阻塞DML。
8. 支持可传输表空间(transportable tablespaces)
9. 支持统计信息的持久化。避免主从之间或数据库重启后，同一个SQL的执行计划有差异
10. 支持支持全文索引
11. 支持InnoDB Memcached plugin
12. EXPLAIN可用来查看DELETE，INSERT，REPLACE，UPDATE等DML操作的执行计划，在此之前，只支持SELECT操作
13. 分区表的增强，包括最大可用分区数增加至8192，支持分区和非分区表之间的数据交换，操作时显式指定分区
14. Redo Log总大小的限制从之前的4G扩展至512G
15. Undo Log可保存在独立表空间中，因其是随机IO，更适合放到SSD中。但仍然不支持空间的自动回收
16. 支持在线和关闭MySQL时dump和load Buffer pool的状态，避免数据库重启后需要较长的预热时间
17. InnoDB内部的性能提升，包括拆分kernel mutex，引入独立的刷新线程，可设置多个purge线程
18. 优化器性能提升，引入了ICP，MRR，BKA等特性，针对子查询进行了优化

MySQL5.7

1. 支持组复制和InnoDB Cluster
2. 支持多源复制
3. 支持增强半同步（AFTER_SYNC）
4. 支持基于表级别(LOGICAL_CLOCK)的并行复制
5. 支持在线开启GTID复制

6. 支持在线设置复制过滤规则
7. 支持在线修改Buffer pool的大小
8. 支持在同一长度编码字节内，修改VARCHAR的大小只需修改表的元数据，无需创建临时表
9. 支持可设置NUMA架构的内存分配策略(innodb_numa_interleave)
10. 支持透明页压缩(Transparent Page Compression)
11. 支持UNDO表空间的自动回收
12. 支持查询优化器的增强和重构
13. 可查看当前session正在执行的SQL的执行计划(EXPLAIN FOR CONNECTION)
14. 引入了查询改写插件（Query Rewrite Plugin），可在服务端对查询进行改写
15. EXPLAIN FORMAT=JSON会显示成本信息，这样可直观的比较两种执行计划的优劣
16. 引入了虚拟列，类似于Oracle中的函数索引
17. 新实例不再默认创建test数据库及匿名用户
18. 引入ALTER USER命令，可用来修改用户密码，密码的过期策略，及锁定用户等
19. mysql.user表中存储密码的字段从password修改为authentication_string
20. 支持表空间加密
21. 优化了Performance Schema，其内存使用减少
22. Performance Schema引入了众多instrumentation。常用的有Memory usage instrumentation，可用来查看MySQL的内存使用情况，Metadata Locking Instrumentation，可用来查看MDL的持有情况，Stage Progress instrumentation，可用来查看Online DDL的进度
23. 同一触发事件（INSERT，DELETE，UPDATE），同一触发时间（BEFORE，AFTER），允许创建多个触发器在此之前，只允许创建一个触发器
24. InnoDB原生支持分区表，在此之前，是通过ha_partition接口来实现的
25. 分区表支持可传输表空间特性。
26. 集成了SYS数据库，简化了MySQL的管理及异常问题的定位
27. 原生支持JSON类型，并引入了众多JSON函数
28. 引入了新的逻辑备份工具mysqlpump，支持表级别的多线程备份
29. 引入了新的客户端工具mysqlsh，其支持三种语言：JavaScript, Python and SQL。两种API：X DevAPI，AdminAPI，其中，前者可将MySQL作为文档型数据库进行操作，后者用于管理InnoDB Cluster
30. mysql_install_db被mysqld --initialize代替，用来进行实例的初始化
31. 原生支持systemd
32. 引入了super_read_only选项
33. 可设置SELECT操作的超时时长（max_execution_time）
34. 可通过SHUTDOWN命令关闭MySQL实例。
35. 引入了innodb_deadlock_detect选项，在高并发场景下，可使用该选项来关闭死锁检测
36. 引入了Optimizer Hints，可在语句级别控制优化器的行为，如是否开启ICP，MRR等，在此之前，只有Index Hints
37. GIS的增强，包括使用Boost.Geometry替代之前的GIS算法，InnoDB开始支持空间索引

MySQL8.0

1. 引入了原生的，基于InnoDB的数据字典。数据字典表位于mysql库中，对用户不可见，同mysql库的其它系统表一样，保存在数据目录下的mysql.ibd文件中。不再置于mysql目录下
2. 重构了INFORMATION_SCHEMA，其中部分表已重构为基于数据字典的视图，在此之前，其为临时表
3. PERFORMANCE_SCHEMA查询性能提升，其已内置多个索引
4. 支持不可见索引(Invisible index)
5. 支持降序索引
6. 优化器加入了直方图功能，对比Oracle
7. 支持公用表表达式(Common table expressions)
8. 支持窗口函数(Window functions)。

9. 支持角色（Role）功能，对比Oracle
10. 支持资源组（Resource Groups），可用来控制线程的优先级及其能使用的资源，目前，能被管理的资源只有CPU
11. 引入了innodb_dedicated_server选项，可基于服务器的内存来动态设置innodb_buffer_pool_size，innodb_log_file_size和innodb_flush_method
12. 支持秒加字段(Instant add column)功能
13. JSON字段的部分更新（JSON Partial Updates）
14. 支持自增主键的持久化
15. 支持可持久化全局变量（SET PERSIST）
16. 默认字符集由latin1修改为utf8mb4
17. 默认开启UNDO表空间，且支持在线调整数量（innodb_undo_tablespaces）。在MySQL 5.7中，默认不开启，若要开启，只能初始化时设置
18. 支持备份锁
19. Redo Log的优化，包括允许多个用户线程并发写入log buffer，可动态修改innodb_log_buffer_size的大小
20. 默认的认证插件由mysql_native_password更改为caching_sha2_password
21. 默认的内存临时表由MEMORY引擎更改为TempTable引擎，相比于前者，后者支持以变长方式存储VARCHAR，VARBINARY等变长字段。从MySQL 8.0.13开始，TempTable引擎支持BLOB字段
22. Grant不再隐式创建用户
23. SELECT ... FOR SHARE和SELECT ... FOR UPDATE语句中引入NOWAIT和SKIP LOCKED选项，解决电商场景热点行问题
24. 正则表达式的增强，新增了4个相关函数，REGEXP_INSTR()，REGEXP_LIKE()，REGEXP_REPLACE()，REGEXP_SUBSTR()
25. 查询优化器在制定执行计划时，会考虑数据是否在Buffer Pool中。而在此之前，是假设数据都在磁盘中
26. ha_partition接口从代码层移除，如果要使用分区表，只能使用InnoDB存储引擎
27. 引入了更多细粒度的权限来替代SUPER权限，现在授予SUPER权限会提示warning
28. GROUP BY语句不再隐式排序
29. information_schema中的innodb_locks和innodb_lock_waits表被移除，取而代之的是performance_schema中的data_locks和data_lock_waits表
30. 引入performance_schema.variables_info表，记录了参数的来源及修改情况
31. 增加了对于客户端报错信息的统计（performance_schema.events_errors_summary_xxx）
32. 可统计查询的响应时间分布（call sys.ps_statement_avg_latency_histogram()）
33. 支持直接修改列名（ALTER TABLE ... RENAME COLUMN old_name TO new_name）
34. 用户密码可设置重试策略（Reuse Policy）
35. 移除PASSWORD()函数。这就意味着无法通过“SET PASSWORD ... = PASSWORD('auth_string') ”命令修改用户密码
36. 代码层移除Query Cache模块，故Query Cache相关的变量和操作均不再支持
37. BLOB, TEXT, GEOMETRY和JSON字段允许设置默认值
38. 可通过RESTART命令重启MySQL实例

22.重做日志(redo log)和二进制日志(binlog)的区别

1. 涉及存储引擎不一样：
    - binlog记录的是所有存储引擎的操作记录
    - redo log只记录innodb存储引擎的日志
2. 记录内容不一样：
    - binlog记录的是关于一个事务的具体操作内容。为逻辑日志
    - 而redo log记录的是每个页更改的物理情况
3. 写的时间不一样：

- binlog文件仅在事务提交前进行提交，即只写磁盘一次
- 而在事务进行过程中，却不断有重做日志条目被写入到重做日志文件中。

23.索引是个什么样的数据结构呢?

索引的数据结构和具体存储引擎的实现有关, 在MySQL中使用较多的索引有Hash索引,B+树索引等,而我们经常使用的InnoDB存储引擎的默认索引实现为:B+树索引

24.Hash索引和B+树所有有什么区别或者说优劣呢?

首先要知道Hash索引和B+树索引的底层实现原理:

hash索引底层就是hash表,进行查找时,调用一次hash函数就可以获取到相应的键值,之后进行回表查询获得实际数据.B+树底层实现是多路平衡查找树.

对于每一次的查询都是从根节点出发,查找到叶子节点方可以获得所查键值,然后根据查询判断是否需要回表查询数据.

那么可以看出他们有以下的不同:

- hash索引进行等值查询更快(一般情况下),但是却无法进行范围查询.

因为在hash索引中经过hash函数建立索引之后,索引的顺序与原顺序无法保持一致,不能支持范围查询.

而B+树的的所有节点皆遵循(左节点小于父节点,右节点大于父节点,多叉树也类似),天然支持范围.

- hash索引不支持使用索引进行排序,原理同上.
- hash索引不支持模糊查询以及多列索引的最左前缀匹配.原理也是因为hash函数的不可预测.**AAAA**和**AAAAB**的索引没有相关性.
- hash索引任何时候都避免不了回表查询数据,而B+树在符合某些条件(聚簇索引,覆盖索引等)的时候可以只通过索引完成查询.
- hash索引虽然在等值查询上较快,但是不稳定.性能不可预测,当某个键值存在大量重复的时候,发生hash碰撞,此时效率可能极差.而B+树的查询效率比较稳定,对于所有的查询都是从根节点到叶子节点,且树的高度较低.

因此,在大多数情况下,直接选择B+树索引可以获得稳定且较好的查询速度.而不需要使用hash索引.

25.上面提到了B+树在满足聚簇索引和覆盖索引的时候不需要回表查询数据,什么是聚簇索引?

- 在B+树的索引中,叶子节点可能存储了当前的key值,也可能存储了当前的key值以及整行的数据,这就是聚簇索引和非聚簇索引.
- 在InnoDB中,只有主键索引是聚簇索引,如果没有主键,则挑选一个唯一键建立聚簇索引.如果没有唯一键,则隐式的生成一个键来建立聚簇索引.
- 当查询使用聚簇索引时,在对应的叶子节点,可以获取到整行数据,因此不用再次进行回表查询.

26.非聚簇索引一定会回表查询吗?

不一定,这涉及到查询语句所要求的字段是否全部命中了索引,如果全部命中了索引,那么就不必再进行回表查询.

举个简单的例子,假设我们在员工表的年龄上建立了索引,那么当进行

```
select age from employee where age < 20
```

查询时,在索引的叶子节点上,已经包含了age信息,不会再次进行回表查询.

27.联合索引是什么?为什么需要注意联合索引中的顺序?

MySQL可以使用多个字段同时建立一个索引,叫做联合索引.在联合索引中,如果想要命中索引,需要按照建立索引时的字段顺序挨个使用,否则无法命中索引.

具体原因为:

- MySQL使用索引时需要索引有序,假设现在建立了"name,age,school"的联合索引
- 那么索引的排序为: 先按照name排序,如果name相同,则按照age排序,如果age的值也相等,则按照school进行排序.
- 当进行查询时,此时索引仅仅按照name严格有序,因此必须首先使用name字段进行等值查询,之后对于匹配到的列而言,其按照age字段严格有序,此时可以使用age字段用做索引查找,以此类推.

因此在建立联合索引的时候应该注意索引列的顺序,一般情况下,将查询需求频繁或者字段选择性高的列放在前面.此外可以根据特例的查询或者表结构进行单独的调整.

28.在哪些情况下会发生针对该列创建了索引但是在查询的时候并没有使用呢?

- 使用不等于查询
- 列参与了数学运算或者函数
- 在字符串like时左边是通配符.类似于'%aaa'.
- 当mysql分析全表扫描比使用索引快的时候不使用索引.
- 当使用联合索引,前面一个条件为范围查询,后面的即使符合最左前缀原则,也无法使用索引.

以上情况,MySQL无法使用索引.

29.同时有多个事务在进行会怎么样呢?

多事务的并发进行一般会造成以下几个问题:

- 脏读: A事务读取到了B事务未提交的内容,而B事务后面进行了回滚.
- 不可重复读: 当设置A事务只能读取B事务已经提交的部分,会造成在A事务内的两次查询,结果竟然不一样,因为在此期间B事务进行了提交操作.
- 幻读: A事务读取了一个范围的内容,而同时B事务在此期间插入了一条数据.造成"幻觉".

30.怎么解决这些问题呢?MySQL的事务隔离级别了解吗?

MySQL的四种隔离级别如下:

- 未提交读(READ UNCOMMITTED)
    - 这就是上面所说的例外情况了,这个隔离级别下,其他事务可以看到本事务没有提交的部分修改.因此会造成脏读的问题(读取到了其他事务未提交的部分,而之后该事务进行了回滚).
    - 这个级别的性能没有足够大的优势,但是又有很多的问题,因此很少使用.

- 已提交读(READ COMMITTED)
    - 其他事务只能读取到本事务已经提交的部分.这个隔离级别有 不可重复读的问题,在同一个事务内的两次读取,拿到的结果竟然不一样,因为另外一个事务对数据进行了修改.
- REPEATABLE READ(可重复读)
    - 可重复读隔离级别解决了上面不可重复读的问题(看名字也知道),但是仍然有一个新问题,就是幻读
    - 当你读取id> 10 的数据行时,对涉及到的所有行加上了读锁,此时例外一个事务新插入了一条id=11的数据,因为是新插入的,所以不会触发上面的锁的排斥
    - 那么进行本事务进行下一次的查询时会发现有一条id=11的数据,而上次的查询操作并没有获取到,再进行插入就会有主键冲突的问题.
- SERIALIZABLE(可串行化)
    - 这是最高的隔离级别,可以解决上面提到的所有问题,因为他强制将所以的操作串行执行,这会导致并发性能极速下降,因此也不是很常用.

31.主键使用自增ID还是UUID?

推荐使用自增ID,不要使用UUID.

- 因为在InnoDB存储引擎中,主键索引是作为聚簇索引存在的
- 也就是说,主键索引的B+树叶子节点上存储了主键索引以及全部的数据(按照顺序)
- 如果主键索引是自增ID,那么只需要不断向后排列即可,如果是UUID,由于到来的ID与原来的大小不确定,会造成非常多的数据插入,数据移动,然后导致产生很多的内存碎片,进而造成插入性能的下降.

总之,在数据量大一些的情况下,用自增主键性能会好一些.

32.MySQL的binlog有有几种录入格式?分别有什么区别?

有三种格式,statement,row和mixed.

- statement模式下,记录单元为语句.即每一个sql造成的影响会记录.由于sql的执行是有上下文的,因此在保存的时候需要保存相关的信息,同时还有一些使用了函数之类的语句无法被记录复制.
- row级别下,记录单元为每一行的改动,基本是可以全部记下来但是由于很多操作,会导致大量行的改动(比如alter table),因此这种模式的文件保存的信息太多,日志量太大.
- mixed. 一种折中的方案,普通操作使用statement记录,当无法使用statement的时候使用row.

33.超大分页怎么处理?

超大的分页一般从两个方向上来解决.

- 数据库层面,这也是我们主要集中关注的(虽然收效没那么大)
    - 类似于`select * from table where age > 20 limit 1000000,10`这种查询其实也是有可以优化
    - 这条语句需要load1000000数据然后基本上全部丢弃,只取10条当然比较慢.
    - 我们可以修改为`select * from table where id in (select id from table where age > 20 limit 1000000,10)`
    - 这样虽然也load了一百万的数据,但是由于索引覆盖,要查询的所有字段都在索引中,所以速度会很快.
    - 同时如果ID连续的好,我们还可以`select * from table where id > 1000000 limit 10`,效率也是不错的

    - 优化的可能性有许多种,但是核心思想都一样,就是减少load的数据.
- 从需求的角度减少这种请求
    - 主要是不做类似的需求(直接跳转到几百万页之后的具体某一页.只允许逐页查看或者按照给定的路线走,这样可预测,可缓存)以及防止ID泄漏且连续被人恶意攻击.

解决超大分页,其实主要是靠缓存,可预测性的提前查到内容,缓存至redis等k-V数据库中,直接返回即可.

34.说一说三个范式

- 第一范式: 每个列都不可以再拆分.
- 第二范式: 非主键列完全依赖于主键,而不能是依赖于主键的一部分.
- 第三范式: 非主键列只依赖于主键,不依赖于其他非主键.

在设计数据库结构的时候,要尽量遵守三范式,如果不遵守,必须有足够的理由.比如性能. 事实上我们经常会为了性能而妥协数据库的设计.

Redis

1.什么是 Redis?

- Redis 是完全开源免费的，遵守 BSD 协议，是一个高性能的 key-value 数据库。
- Redis 与其他 key - value 缓存产品有以下三个特点
    1. Redis 支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。
    2. Redis 不仅仅支持简单的 key-value 类型的数据，同时还提供 list，set，zset，hash 等数据结构的存储。
    3. Redis 支持数据的备份，即 master-slave 模式的数据备份。

Redis 优势

- 性能极高 – Redis 能读的速度是 110000 次/s,写的速度是 81000 次/s 。
- 丰富的数据类型 – Redis 支持二进制案例的 Strings, Lists, Hashes, Sets 及Ordered Sets 数据类型操作。
- 原子 – Redis 的所有操作都是原子性的，意思就是要么成功执行要么失败完全不执行。单个操作是原子性的。多个操作也支持事务，即原子性，通过 MULTI 和 EXEC指令包起来。
- 丰富的特性 – Redis 还支持 publish/subscribe, 通知, key 过期等等特性。

Redis 与其他 key-value 存储有什么不同?

- Redis 有着更为复杂的数据结构并且提供对他们的原子性操作，这是一个不同于其他数据库的进化路径。Redis 的数据类型都是基于基本数据结构的同时对程序员透明，无需进行额外的抽象。
- Redis 运行在内存中但是可以持久化到磁盘，所以在对不同数据集进行高速读写时需要权衡内存，因为数据量不能大于硬件内存。在内存数据库方面的另一个优点是，相比在磁盘上相同的复杂的数据结构，在内存中操作起来非常简单，这样 Redis可以做很多内部复杂性很强的事情。同时，在磁盘格式方面他们是紧凑的以追加的方式产生的，因为他们并不需要进行随机访问。

2.Redis 的数据类型?

- Redis 支持五种数据类型：string（字符串），hash（哈希），list（列表），set（集合）及 zsetsorted set：有序集合)。
- 我们实际项目中比较常用的是 string，hash 如果你是 Redis 中高级用户，还需要加上下面几种数据结构 HyperLogLog、Geo、Pub/Sub。
- 如果你说还玩过 Redis Module，像 BloomFilter，RedisSearch，Redis-ML，面试官得眼睛就开始发亮了。

3.使用 Redis 有哪些好处?

- 速度快，因为数据存在内存中，类似于 HashMap，HashMap 的优势就是查找和操作的时间复杂度都是 O1)
- 支持丰富数据类型，支持 string，list，set，Zset，hash 等
- 支持事务，操作都是原子性，所谓的原子性就是对数据的更改要么全部执行，要么全部不执行
- 丰富的特性：可用于缓存，消息，按 key 设置过期时间，过期后将会自动删除

4.Redis 相比 Memcached 有哪些优势?

- Memcached 所有的值均是简单的字符串，redis 作为其替代者，支持更为丰富的数据类
- Redis 的速度比 Memcached 快很
- Redis 可以持久化其数据

变种问题: Memcache 与 Redis 的区别都有哪些?

- 存储方式 Memecache 把数据全部存在内存之中，断电后会挂掉，数据不能超过内存大小。 Redis 有部份存在硬盘上，这样能保证数据的持久性。
- 数据支持类型 Memcache 对数据类型支持相对简单。 Redis 有复杂的数据类型。
- 使用底层模型不同 它们之间底层实现方式 以及与客户端之间通信的应用协议不一样。 Redis 直接自己构建了 VM 机制 ，因为一般的系统调用系统函数的话，会浪费一定的时间去移动和请求。

5.Redis 是单进程单线程的?

Redis 是单进程单线程的，redis 利用队列技术将并发访问变为串行访问，消除了传统数据库串行控制的开销。

6.一个字符串类型的值能存储最大容量是多少?

- 512M

7.Redis 的持久化机制是什么？各自的优缺点?

- Redis提供两种持久化机制 RDB 和 AOF 机制

1.RDBRedis DataBase 持久化方式

- 是指用数据集快照的方式半持久化模式)记录 redis 数据库的所有键值对,在某个时间点将数据写入一个临时文件，持久化结束后，用这个临时文件替换上次持久化的文件，达到数据恢复。
- 优点：
    1. 只有一个文件 dump.rdb，方便持久化。

2. 容灾性好，一个文件可以保存到安全的磁盘。
3. 性能最大化，fork 子进程来完成写操作，让主进程继续处理命令，所以是 IO最大化。使用单独子进程来进行持久化，主进程不会进行任何 IO 操作，保证了 redis的高性能)
4. 相对于数据集大时，比 AOF 的启动效率更高。

- 缺点：
  - 数据安全性低。RDB 是间隔一段时间进行持久化，如果持久化之间 redis 发生故障，会发生数据丢失。所以这种方式更适合数据要求不严谨的时候

2.AOFAppend-only file)持久化方式：

- 是指所有的命令行记录以 redis 命令请求协议的格式完全持久化存储)保存为 aof 文件。
- 优点：
  1. 数据安全，aof 持久化可以配置 appendfsync 属性，有 always，每进行一次命令操作就记录到 aof 文件中一次。
  2. 通过 append 模式写文件，即使中途服务器宕机，可以通过 redis-check-aof工具解决数据一致性问题。
  3. AOF 机制的 rewrite 模式。AOF 文件没被 rewrite 之前（文件过大时会对命令进行合并重写），可以删除其中的某些命令（比如误操作的 flushall））
- 缺点：
  1. AOF 文件比 RDB 文件大，且恢复速度慢。
  2. 数据集大的时候，比 rdb 启动效率低。

8.Redis 常见性能问题和解决方案：

1. Master 最好不要写内存快照，如果 Master 写内存快照，save 命令调度 rdbSave函数，会阻塞主线程的工作，当快照比较大时对性能影响是非常大的，会间断性暂停服务
2. 如果数据比较重要，某个 Slave 开启 AOF 备份数据，策略设置为每秒同步
3. 为了主从复制的速度和连接的稳定性，Master 和 Slave 最好在同一个局域网
4. 尽量避免在压力很大的主库上增加从
5. 主从复制不要用图状结构，用单向链表结构更为稳定，即：Master <- Slave1<- Slave2 <- Slave3… 这样的结构方便解决单点故障问题，实现 Slave 对 Master的替换。如果 Master 挂了，可以立刻启用 Slave1 做 Master，其他不变。

9.redis 过期键的删除策略?

- 定时删除
  - 在设置键的过期时间的同时，创建一个定时器 timer). 让定时器在键的过期时间来临时，立即执行对键的删除操作。
- 惰性删除
  - 放任键过期不管，但是每次从键空间中获取键时，都检查取得的键是否过期，如果过期的话，就删除该键;如果没有过期，就返回该键。
- 定期删除
  - 每隔一段时间程序就对数据库进行一次检查，删除里面的过期键。至于要删除多少过期键，以及要检查多少个数据库，则由算法决定。

10.Redis 的回收策略（淘汰策略）？

1. volatile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰
2. volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰
3. volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰
4. allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰
5. allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰
6. no-enviction（驱逐）：禁止驱逐数据

注意这里的 6 种机制，volatile 和 allkeys 规定了是对已设置过期时间的数据集淘汰数据还是从全部数据集淘汰数据，后面的 lru、ttl 以及 random 是三种不同的淘汰策略，再加上一种 no-enviction 永不回收的策略。

使用策略规则：

- 如果数据呈现幂律分布，也就是一部分数据访问频率高，一部分数据访问频率低，则使用 allkeys-lru
- 如果数据呈现平等分布，也就是所有的数据访问频率都相同，则使用allkeys-random

变种问题1: Redis 的内存用完了会发生什么?

- 如果达到设置的上限，Redis 的写命令会返回错误信息(但是读命令还可以正常返回)。或者你可以将 Redis 当缓存来使用配置淘汰机制，当 Redis 达到内存上限时会冲刷掉旧的内容。

变种问题2: MySQL 里有 2000w 数据，redis 中只存 20w 的数据，如何保证 redis 中的数据都是热点数据?

- Redis 内存数据集大小上升到一定大小的时候，就会施行数据淘汰策略。

11.为什么 redis 需要把所有数据放到内存中?

- Redis 为了达到最快的读写速度将数据都读到内存中，并通过异步的方式将数据写入磁盘。所以 redis 具有快速和数据持久化的特征。如果不将数据放在内存中，磁盘 I/O 速度为严重影响 redis 的性能。在内存越来越便宜的今天，redis 将会越来越受欢迎。如果设置了最大使用的内存，则数据已有记录数达到内存限值后不能继续插入新值。

12.Redis 的同步机制了解么?

Redis 可以使用主从同步，从从同步。

- 第一次同步时，主节点做一次 bgsave，并同时将后续修改操作记录到内存 buffer，待完成后将 rdb 文件全量同步到复制节点，复制节点接受完成后将 rdb 镜像加载到内存。
- 加载完成后，再通知主节点将期间修改的操作记录同步到复制节点进行重放就完成了同步过程。

13.Pipeline 有什么好处，为什么要用 pipeline?

- 可以将多次 IO 往返的时间缩减为一次，前提是 pipeline 执行的指令之间没有因果相关性。
- 使用 redis-benchmark 进行压测的时候可以发现影响 redis 的 QPS峰值的一个重要因素是 pipeline 批次指令的数目。

14.是否使用过 Redis 集群，集群的原理是什么?

1.主从模式

主从模式是三种模式中最简单的，在主从复制中，数据库分为两类：主数据库(master)和从数据库(slave)。

- 其中主从复制有如下特点
  - 主数据库可以进行读写操作，当读写操作导致数据变化时会自动将数据同步给从数据库
  - 从数据库一般都是只读的，并且接收主数据库同步过来的数据
  - 一个master可以拥有多个slave，但是一个slave只能对应一个master
  - slave挂了不影响其他slave的读和master的读和写，重新启动后会将数据从master同步过来
  - master挂了以后，不影响slave的读，但redis不再提供写服务，master重启后redis将重新对外提供写服务
  - master挂了以后，不会在slave节点中重新选一个master
- 工作机制
  - 当slave启动后，主动向master发送SYNC命令。master接收到SYNC命令后在后台保存快照（RDB持久化）和缓存保存快照这段时间的命令，然后将保存的快照文件和缓存的命令发送给slave。slave接收到快照文件和命令后加载快照文件和缓存的执行命令。
  - 复制初始化后，master每次接收到的写命令都会同步发送给slave，保证主从数据一致性。
- 缺点
  - 从上面可以看出，master节点在主从模式中唯一，若master挂掉，则redis无法对外提供写服务。

2.Sentinel模式

主从模式的弊端就是不具备高可用性，当master挂掉以后，Redis将不能再对外提供写入操作，因此sentinel应运而生。

- sentinel中文含义为哨兵，顾名思义，它的作用就是监控redis集群的运行状况，特点如下
  - sentinel模式是建立在主从模式的基础上，如果只有一个Redis节点，sentinel就没有任何意义
  - 当master挂了以后，sentinel会在slave中选择一个做为master，并修改它们的配置文件，其他slave的配置文件也会被修改，比如slaveof属性会指向新的master
  - 当master重新启动后，它将不再是master而是做为slave接收新的master的同步数据
  - sentinel因为也是一个进程有挂掉的可能，所以sentinel也会启动多个形成一个sentinel集群
  - 多sentinel配置的时候，sentinel之间也会自动监控
  - 当主从模式配置密码时，sentinel也会同步将配置信息修改到配置文件中，不需要担心
  - 一个sentinel或sentinel集群可以管理多个主从Redis，多个sentinel也可以监控同一个redis
  - sentinel最好不要和Redis部署在同一台机器，不然Redis的服务器挂了以后，sentinel也挂了
- 工作机制
  - 每个sentinel以每秒钟一次的频率向它所知的master，slave以及其他sentinel实例发送一个 PING 命令
  - 如果一个实例距离最后一次有效回复 PING 命令的时间超过 down-after-milliseconds 选项所指定的值， 则这个实例会被sentinel标记为主观下线。
  - 如果一个master被标记为主观下线，则正在监视这个master的所有sentinel要以每秒一次的频率确认master的确进入了主观下线状态
  - 当有足够数量的sentinel（大于等于配置文件指定的值）在指定的时间范围内确认master的确进入了主观下线状态， 则master会被标记为客观下线
  - 在一般情况下， 每个sentinel会以每 10 秒一次的频率向它已知的所有master，slave发送 INFO 命令
  - 当master被sentinel标记为客观下线时，sentinel向下线的master的所有slave发送 INFO 命令的频率会从 10 秒一次改为 1 秒一次
  - 若没有足够数量的sentinel同意master已经下线，master的客观下线状态就会被移除

- 若master重新向sentinel的 PING 命令返回有效回复，master的主观下线状态就会被移除

补充

当使用sentinel模式的时候，客户端就不要直接连接Redis，而是连接sentinel的ip和port，由sentinel来提供具体的可提供服务的Redis实现，这样当master节点挂掉以后，sentinel就会感知并将新的master节点提供给使用者。(需要有开发配合)

```
/**
 * 测试Redis哨兵模式
 */
public class TestSentinels {
    @SuppressWarnings("resource")
    @Test
    public void testSentinel() {
        JedisPoolConfig jedisPoolConfig = new JedisPoolConfig();
        jedisPoolConfig.setMaxTotal(10);
        jedisPoolConfig.setMaxIdle(5);
        jedisPoolConfig.setMinIdle(5);
        // 哨兵信息
        Set<String> sentinels = new HashSet<>
(Arrays.asList("192.168.11.128:26379",
                "192.168.11.129:26379","192.168.11.130:26379"));
        // 创建连接池
        JedisSentinelPool pool = new JedisSentinelPool("mymaster",
sentinels,jedisPoolConfig,"123456");
        // 获取客户端
        Jedis jedis = pool.getResource();
        // 执行两个命令
        jedis.set("mykey", "myvalue");
        String value = jedis.get("mykey");
        System.out.println(value);
    }
}
```

3.Cluster模式

- sentinel模式基本可以满足一般生产的需求，具备高可用性。但是当数据量过大到一台服务器存放不下的情况时，主从模式或sentinel模式就不能满足需求了，这个时候需要对存储的数据进行分片，将数据存储到多个Redis实例中。cluster模式的出现就是为了解决单机Redis容量有限的问题，将Redis的数据根据一定的规则分配到多台机器。
- cluster可以说是sentinel和主从模式的结合体，通过cluster可以实现主从和master重选功能，所以如果配置两个副本三个分片的话，就需要六个Redis实例。因为Redis的数据是根据一定规则分配到cluster的不同机器的，当数据量过大时，可以新增机器进行扩容。
- 使用集群，只需要将redis配置文件中的cluster-enable配置打开即可。每个集群中至少需要三个主数据库才能正常运行，新增节点非常方便。
- cluster集群特点：
    - 多个redis节点网络互联，数据共享
    - 所有的节点都是一主一从（也可以是一主多从），其中从不提供服务，仅作为备用
    - 不支持同时处理多个key（如MSET/MGET），因为redis需要把key均匀分布在各个节点上，并发量很高的情况下同时创建key-value会降低性能并导致不可预测的行为
    - 支持在线增加、删除节点
    - 客户端可以连接任何一个主节点进行读写

15.Redis 集群方案什么情况下会导致整个集群不可用?

- 有 A，B，C 三个节点的集群,在没有复制模型的情况下,如果节点 B 失败了，那么整个集群就会以为缺少 5501-11000 这个范围的槽而不可用。

16.Redis 支持的 Java 客户端都有哪些？官方推荐用哪个?

- Redisson、Jedis、lettuce 等等，官方推荐使用 Redisson。

17.Jedis 与 Redisson 对比有什么优缺点?

- Jedis 是 Redis 的 Java 实现的客户端，其 API 提供了比较全面的 Redis 命令的支持
- Redisson 实现了分布式和可扩展的 Java 数据结构，和 Jedis 相比，功能较为简单，不支持字符串操作，不支持排序、事务、管道、分区等 Redis 特性。
- Redisson 的宗旨是促进使用者对 Redis 的关注分离，从而让使用者能够将精力更集中地放在处理业务逻辑上。

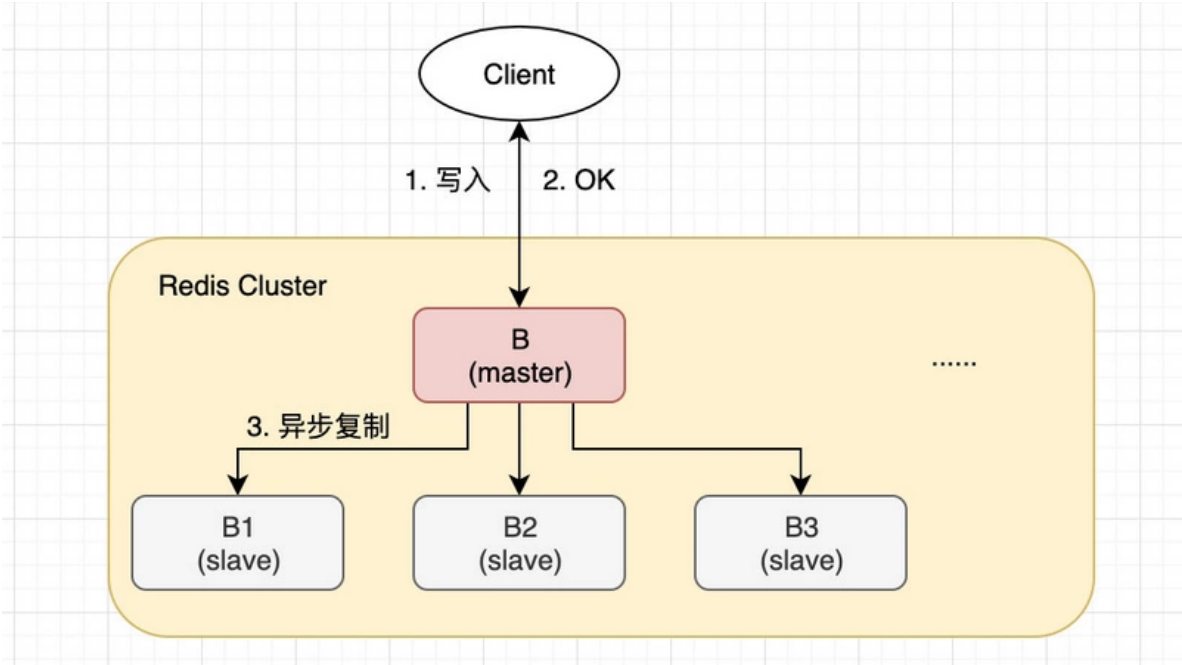
18.说说 Redis 哈希槽的概念?

- Redis 集群没有使用一致性 hash,而是引入了哈希槽的概念，Redis 集群有16384 个哈希槽，每个 key 通过 CRC16 校验后对 16384 取模来决定放置哪个槽，集群的每个节点负责一部分 hash 槽。

19.Redis 集群会有写操作丢失吗？为什么?

Redis Cluster 不保证强一致性，在一些特殊场景，客户端即使收到了写入确认，还是可能丢数据的。

场景1：异步复制



- client 写入 master B

- master B 回复 OK
- master B 同步至 slave B1 B2 B3

B 没有等待 B1 B2 B3 的确认就回复了 client，如果在 slave 同步完成之前，master 宕机了，其中一个 slave 会被选为 master，这时之前 client 写入的数据就丢了。

`wait` 命令可以增强这种场景的数据安全性。

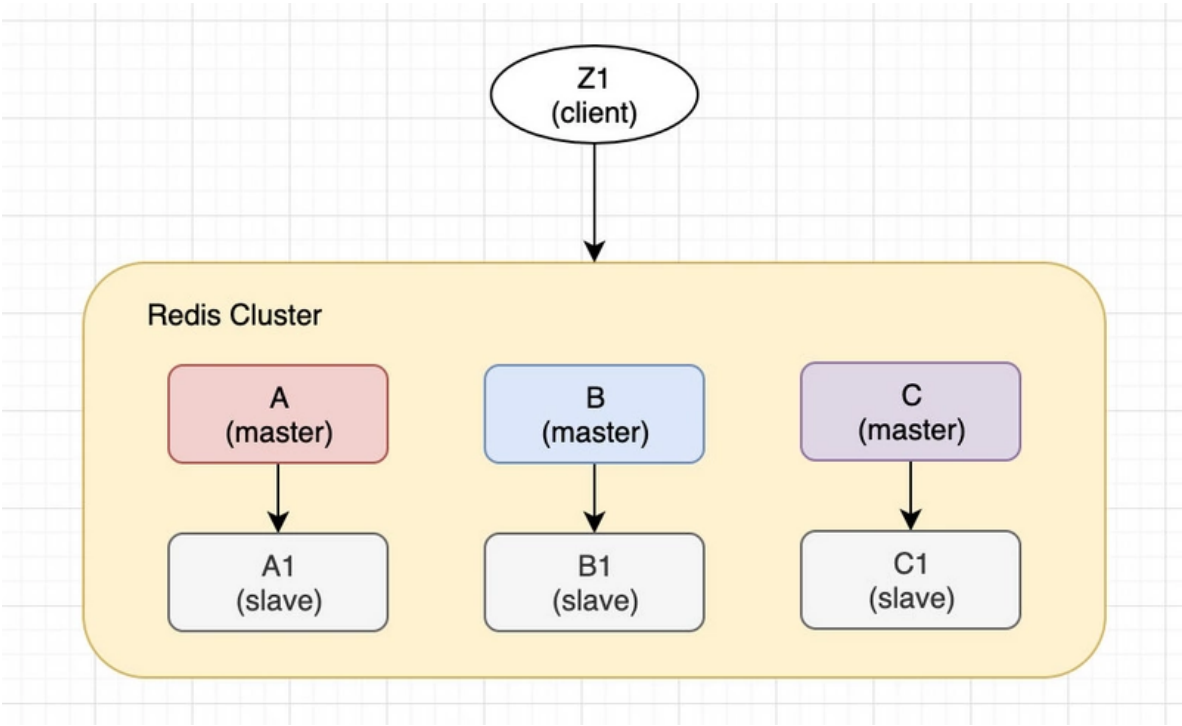
`wait` 会阻塞当前 client 直到之前的写操作被指定数量的 slave 同步成功。

`wait` 可以提高数据的安全性，但并不保证强一致性。

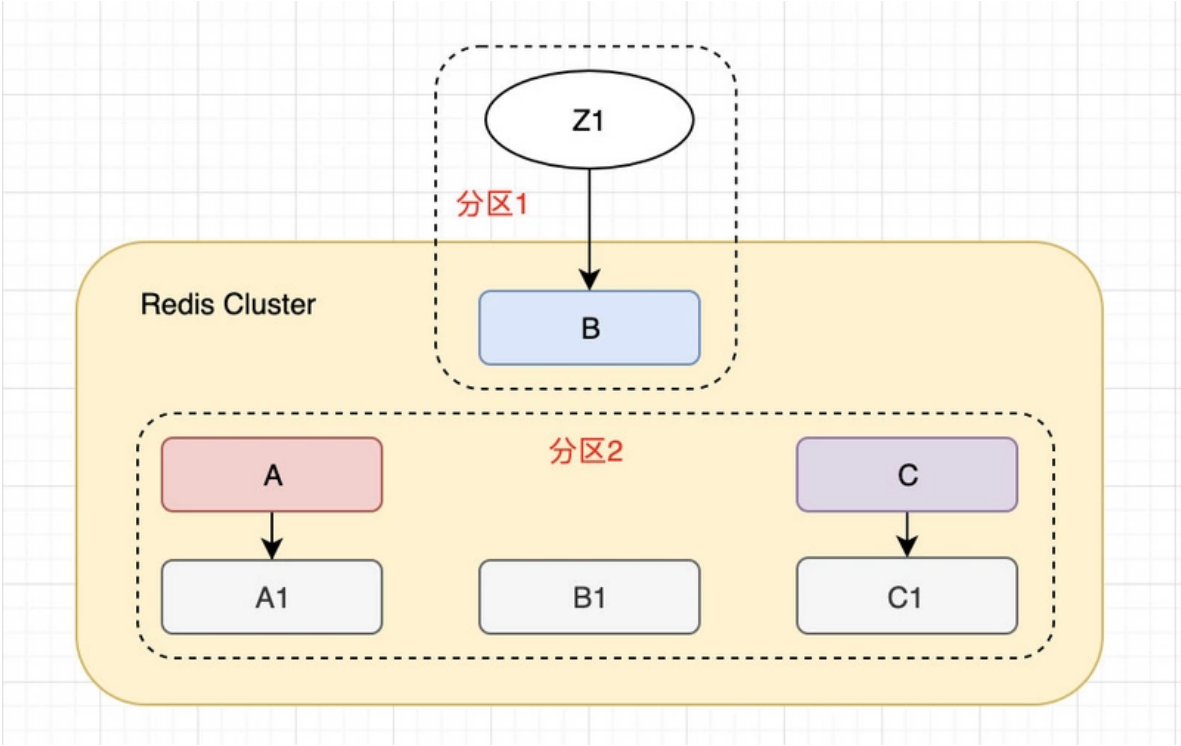
因为即使使用了这种同步复制方式，也存在特殊情况：一个没有完成同步的 slave 被选举为了 master。

场景2：网络分区

6个节点 A, B, C, A1, B1, C1，3个master，3个slave，还有一个client，Z1。



发生网络分区后， 形成了2个区， A, C, A1, B1, C1和B Z1



- 这时 Z1 还是可以向 B 写入的，如果短时间内分区就恢复了，那就没问题，整个集群继续正常工作，但如果时间一长，B1 就会成为所在分区的 master，Z1 写入 B 的数据就丢了。
- maximum window（最大时间窗口） 可以减少数据损失，可以控制 Z1 向 B 写入的总数
  - 过去一定时间后，分区的多数边就会进行选举，slave 成为 master，这时分区少数边的 master 就会拒绝接收写请求。
- 这个**时间量**是非常重要的，称为节点**过期时间**。
- 一个 master 在达到过期时间后，就被认为是故障的，进入 error 状态，停止接收写请求，可以被 slave 取代。

小结

- Redis Cluster 不保证强一致性，存在丢失数据的场景
  - 异步复制
    - 在 master 写成功，但 slave 同步完成之前，master 宕机了，slave 变为 master，数据丢失。
    - wait 命令可以给为同步复制，但也无法完全保证数据不丢，而且影响性能。
  - 网络分区
    - 分区后一个 master 继续接收写请求，分区恢复后这个 master 可能会变为 slave，那么之前写入的数据就丢了。
    - 可以设置节点过期时间，减少 master 在分区期间接收的写入数量，降低数据丢失的损失。

20.怎么理解 Redis 事务?

Redis 事务的本质是一组命令的集合。事务支持一次执行多个命令，一个事务中所有命令都会被序列化。在事务执行过程，会按照顺序串行化执行队列中的命令，其他客户端提交的命令请求不会插入到事务执行命令序列中。

- 没有隔离级别的概念

    - 批量操作在发送 EXEC 命令前被放入队列缓存，并不会被实际执行，也就不存在事务内的查询要看到事务里的更新，事务外查询不能看到。
- 不保证原子性
    - Redis中，单条命令是原子性执行的，但事务不保证原子性，且没有回滚。事务中任意命令执行失败，其余的命令仍会被执行。

21.Redis 如何做内存优化?

- 尽可能使用散列表（hashes），散列表（是说散列表里面存储的数少）使用的内存非常小
- 尽可能的将数据模型抽象到一个散列表里面。
- 比如 web 系统中有一个用户对象，不要为这个用户的名称，姓氏，邮箱，密码设置单独的 key,而是应该把这个用户的所有信息存储到一张散列表里面。

22.Redis 回收进程如何工作的?

- 一个客户端运行了新的命令，添加了新的数据。Redi 检查内存使用情况，如果大于 maxmemory 的限制, 则根据设定好的策略进行回收。
- 所以我们不断地穿越内存限制的边界，通过不断达到边界然后不断地回收回到边界以下。如果一个命令的结果导致大量内存被使用（例如很大的集合的交集保存到一个新的键），不用多久内存限制就会被这个内存使用量超越。

23.Redis 最适合的场景?

1. **会话缓存**（Session Cache）

最常用的一种使用 Redis 的情景是会话缓存（session cache）。

用 Redis 缓存会话比其他存储（如 Memcached）的优势在于：Redis 提供持久化。当维护一个不是严格要求一致性的缓存时，如果用户的购物车信息全部丢失，大部分人都会不高兴的，现在，他们还会这样吗？ 幸运的是，随着 Redis 这些年的改进，很容易找到怎么恰当的使用 Redis 来缓存会话的文档。甚至广为人知的商业平台Magento 也提供 Redis 的插件。

2. **全页缓存**（FPC）

除基本的会话 token 之外，Redis 还提供很简便的 FPC 平台。回到一致性问题，即使重启了 Redis 实例，因为有磁盘的持久化，用户也不会看到页面加载速度的下降，这是一个极大改进，类似 PHP 本地 FPC。 再次以 Magento 为例，Magento提供一个插件来使用 Redis 作为全页缓存后端。 此外，对 WordPress 的用户来说，Pantheon 有一个非常好的插件 wp-redis，这个插件能帮助你以最快速度加载你曾浏览过的页面。

3. **队列**

Reids 在内存存储引擎领域的一大优点是提供 list 和 set 操作，这使得 Redis能作为一个很好的消息队列平台来使用。Redis 作为队列使用的操作，就类似于本地程序语言（如 Python）对 list 的 push/pop 操作。 如果你快速的在 Google中搜索“Redis queues”，你马上就能找到大量的开源项目，这些项目的目的就是利用 Redis 创建非常好的后端工具，以满足各种队列需求。例如，Celery 有一个后台就是使用 Redis 作为 broker，你可以从这里去查看。

4. **排行榜/计数器**

Redis 在内存中对数字进行递增或递减的操作实现的非常好。集合（Set）和有序集合（Sorted Set）也使得我们在执行这些操作的时候变的非常简单，Redis 只是正好提供了这两种数据结构。所以，我们要从排序集合中获取到排名最靠前的 10个用户–我们称之为“user_scores”，我们只需要像下面一样执行即可： 当然，这是假定你是根据你用户的分数做递增的排序。如果你想返回用户及用户的分数，你需要这样执行： ZRANGE user_scores 0 10 WITHSCORES Agora Games 就是一个很好的例子，用 Ruby 实现的，它的排行榜就是使用 Redis 来存储数据的，你可以在这里看到。

5. **发布/订阅**

最后（但肯定不是最不重要的）是 Redis 的发布/订阅功能。发布/订阅的使用场景确实非常多。我已看见人们在社交网络连接中使用，还可作为基于发布/订阅的脚本触发器，甚至用 Redis 的发布/订阅功能来建立聊天系统！

24. 假如 Redis 里面有 1 亿个 key，其中有 10w 个 key 是以某个固定的已知的前缀开头的，如果将它们全部找出来?

- 使用 keys 指令可以扫出指定模式的 key 列表。

追问：如果这个 redis 正在给线上的业务提供服务，那使用 keys 指令会有什么问题?

- 这个时候你要回答 redis 关键的一个特性：redis 的单线程的。
- keys 指令会导致线程阻塞一段时间，线上服务会停顿，直到指令执行完毕，服务才能恢复。
- 这个时候可以使用 scan 指令，scan 指令可以无阻塞的提取出指定模式的 key 列表，但是会有一定的重复概率，在客户端做一次去重就可以了，但是整体所花费的时间会比直接用 keys 指令长。

25.使用过 Redis 做异步队列么，你是怎么用的?

一般使用 list 结构作为队列，rpush 生产消息，lpop 消费消息。当 lpop 没有消息的时候，要适当 sleep 一会再重试。如果对方追问可不可以不用 sleep 呢？list 还有个指令叫 blpop，在没有消息的时候，它会阻塞住直到消息到来。如果对方追问能不能生产一次消费多次呢？使用 pub/sub 主题订阅者模式，可以实现1:N 的消息队列。

如果对方追问 pub/sub 有什么缺点?

在消费者下线的情况下，生产的消息会丢失，得使用专业的消息队列如 RabbitMQ等。

如果对方追问 redis 如何实现延时队列?

我估计现在你很想把面试官一棒打死如果你手上有一根棒球棍的话，怎么问的这么详细。但是你很克制，然后神态自若的回答道：使用 sortedset，拿时间戳作为score，消息内容作为 key 调用 zadd 来生产消息，消费者用 zrangebyscore 指令获取 N 秒之前的数据轮询进行处理。到这里，面试官暗地里已经对你竖起了大拇指。但是他不知道的是此刻你却竖起了中指，在椅子背后。

26.使用过 Redis 分布式锁么，它是什么回事?

先拿 setnx 来争抢锁，抢到之后，再用 expire 给锁加一个过期时间防止锁忘记了释放。

这时候对方会告诉你说你回答得不错，然后接着问如果在 setnx 之后执行 expire之前进程意外 crash 或者要重启维护了，那会怎么样？这时候你要给予惊讶的反馈：唉，是喔，这个锁就永远得不到释放了。紧接着你需要抓一抓自己得脑袋，故作思考片刻，好像接下来的结果是你主动思考出来的，然后回答：我记得 set 指令有非常复杂的参数，这个应该是可以同时把 setnx 和expire 合成一条指令来用的！对方这时会显露笑容，心里开始默念：摁，这小子还不错。

27.Redis雪崩了解么?

目前电商首页以及热点数据都会去做缓存，一般缓存都是定时任务去刷新，或者是查不到之后去更新的，定时任务刷新就有一个问题。

- 举个简单的例子
    - 如果所有首页的Key失效时间都是12小时，中午12点刷新的，我零点有个秒杀活动大量用户涌入，假设当时每秒 6000 个请求，本来缓存在可以扛住每秒 5000 个请求，但是缓存当时所有的Key都失效了。
    - 此时 1 秒 6000 个请求全部落数据库，数据库必然扛不住，它会报一下警，真实情况可能DBA都没反应过来就直接挂了。此时，
    - 如果没用什么特别的方案来处理这个故障，DBA 很着急，重启数据库，但是数据库立马又被新的流量给打死了。
    - 这就是我理解的缓存雪崩。

我刻意看了下我做过的项目感觉再吊的都不允许这么大的QPS直接打DB去，不过没慢SQL加上分库，大表分表可能还还算能顶，但是跟用了Redis的差距还是很大。

同一时间大面积失效，那一瞬间Redis跟没有一样，那这个数量级别的请求直接打到数据库几乎是灾难性的，你想想如果打挂的是一个用户服务的库，那其他依赖他的库所有的接口几乎都会报错，如果没做熔断等策略基本上就是瞬间挂一片的节奏，你怎么重启用户都会把你打挂，等你能重启的时候，用户早就睡觉去了，并且对你的产品失去了信心，什么垃圾产品。

进阶问题: 如果发生了Redis雪崩, 该如何处理?

- 处理缓存雪崩简单，在批量往Redis存数据的时候，把每个Key的失效时间都加个随机值就好了，这样可以保证数据不会在同一时间大面积失效，我相信，Redis这点流量还是顶得住的。

```
setRedis (Key, value, time + Math.random() * 10000)
```

- 如果Redis是集群部署，将热点数据均匀分布在不同的Redis库中也能避免全部失效的问题，不过本渣我在生产环境中操作集群的时候，单个服务都是对应的单个Redis分片，是为了方便数据的管理，但是也同样有了可能会失效这样的弊端，失效时间随机是个好策略。
- 或者设置热点数据永远不过期，有更新操作就更新缓存就好了（比如运维更新了首页商品，那你刷下缓存就完事了，不要设置过期时间），电商首页的数据也可以用这个操作，保险。

28.缓存穿透和击穿是什么? 和雪崩有什么区别

缓存穿透

- 缓存和数据库中都没有的数据，而用户不断发起请求，我们数据库的 id 都是1开始自增上去的，如发起为id值为 -1 的数据或 id 为特别大不存在的数据。这时的用户很可能是攻击者，攻击会导致数据库压力过大，严重会击垮数据库。
- 小点的单机系统，基本上用postman就能搞死，比如我自己买的阿里云服务。
- 像这种如果不对参数做校验，数据库id都是大于0的，攻击方一直用小于0的参数去请求你，每次都能绕开Redis直接打到数据库，数据库也查不到，每次都这样，并发高点就容易崩掉了。

缓存击穿

- 这个跟缓存雪崩有点像，但是又有一点不一样，缓存雪崩是因为大面积的缓存失效，打崩了DB，而缓存击穿不同的是缓存击穿是指一个Key非常热点，在不停的扛着大并发，大并发集中对这一个

点进行访问，当这个Key在失效的瞬间，持续的大并发就穿破缓存，直接请求数据库，就像在一个完好无损的桶上凿开了一个洞。

进阶问题: 分别怎么解决

- 缓存穿透我会在接口层增加校验，比如用户鉴权校验，参数做校验，不合法的参数直接代码Return
    - 比如：id 做基础校验，id <=0的直接拦截等。
- 这里我想提的一点就是，我们在开发程序的时候都要有一颗“不信任”的心，就是不要相信任何调用方，比如你提供了API接口出去，你有这几个参数，那我觉得作为被调用方，任何可能的参数情况都应该被考虑到，做校验，因为你不相信调用你的人，你不知道他会传什么参数给你。
    - 举个简单的例子，你这个接口是分页查询的，但是你没对分页参数的大小做限制，调用的人万一一口气查 Integer.MAX_VALUE 一次请求就要你几秒，多几个并发你不就挂了么？是公司同事调用还好大不了发现了改掉，但是如果是黑客或者竞争对手呢？在你双十一当天就调你这个接口会发生什么，就不用我说了吧。这是之前的Leader跟我说的，我觉得大家也都应该了解下。
- 从缓存取不到的数据，在数据库中也没有取到，这时也可以将对应Key的Value对写为null、位置错误、稍后重试这样的值具体取啥问产品，或者看具体的场景，缓存有效时间可以设置短点，如30秒（设置太长会导致正常情况也没法使用）。
- 这样可以防止攻击用户反复用同一个id暴力攻击，但是我们要知道正常用户是不会在单秒内发起这么多次请求的，那网关层Nginx本渣我也记得有配置项，可以让运维大大对单个IP每秒访问次数超出阈值的IP都拉黑。
- Redis高级用法布隆过滤器（Bloom Filter）这个也能很好的防止缓存穿透的发生，他的原理也很简单就是利用高效的数据结构和算法快速判断出你这个Key是否在数据库中存在，不存在你return就好了，存在你就去查了DB刷新KV再return。
- 如果黑客有很多个IP同时发起攻击呢？这点我一直也不是很想得通，但是一般级别的黑客没这么多肉鸡，再者正常级别的Redis集群都能抗住这种级别的访问的，小公司我想他们不会感兴趣的。把系统的高可用做好了，集群还是很能顶的。
- 缓存击穿的话，设置热点数据永远不过期。或者加上互斥锁就能搞定了