题目一：

一、业务场景介绍：

小猛同学正在压测，发现个小问题，因为在终端设备上跟鹅厂有紧密合作，调用他们的接口时需要获取到access\_token，但是这个access\_token过期时间是2小时，过期后需要重新获取。

压测时发现当到达过期时间时，日志看刷出来好几个不一样的access\_token，因为这个服务也是分布式部署的，多个节点同时发起了第三方接口请求导致。

虽然以最后一次获取的access\_token为准，也没什么不良副作用，但是会导致多次不必要的对第三方接口的调用，也会短时间内造成access\_token的 重复无效获取

问题：请大家思考一下，通过什么技术解决上面的业务场景？



题目二：

一、业务场景

小猛同学在某公司的大数据部门，现在要做一个高并发的同步系统，把某个业务的数据同步到自己的系统中，然后在做各种复杂的操作后提供接口给其他部门查询，每天日同步数据要达到上千万级别。

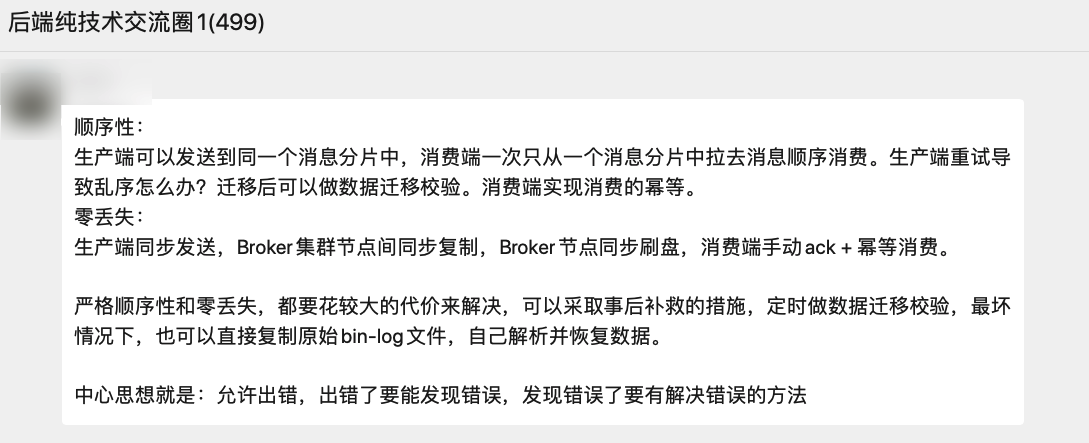
整个数据同步前期流程是从一个 mysql 库原封不动地同步到另一个 mysql 库里面去（mysql -> mysql）。

小猛同学初步架构设计是：从一个mysql监听到binlog后发送消息到MQ中，另外一个服务在消费MQ中的数据写到自己的库中，但是现在存在一个问题。

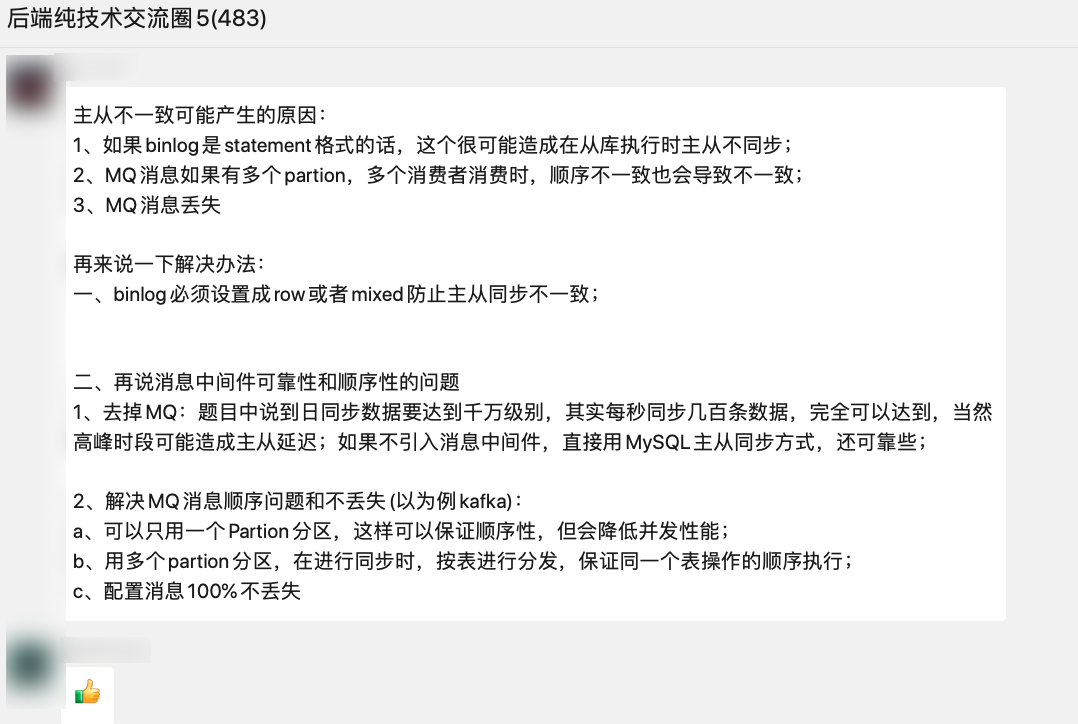
多个业务系统执行sql往mysql 里增删改对应的数据，但是小猛发现自己这边库的数据有时候跟业务方的库数据不一致。

二、大家思考上面问题产生原因，如何去解决上面的技术问题？

答案1：



答案2：



其他汇总方案：

1、小猛同学是使用的RocketMQ，在监听binlog的服务中发送消息到RocketMQ的时候，一条数据的增删改路由到同一个messagequeue中保证消息的顺序性；

2、A同学会问：如何解决监听到binlog后发送消息到RocketMQ的失败问题呢？

答：发送消息时重试、多次重试失败后，提供兜底补偿方案

3、B同学会问：发送binlog01、binlog02，01发送失败重试，02发送成功到时无序怎么办？答：修改MQ的配置，需要等消息01发送成功后才能发送02；或者修改binlog为row模式，不使用insert update delete等逻辑上的binlog（即statement模式）

4、C同学会问：如何保证MQ中的消息不丢失？

答：RocketMQ的master、slave高可用，基于一致性算法raft来写commitlog

5、D同学会问：如何保证消费messagequeue时消息的有序性？

答：单线程消费；

如何提升系统性能？

答：单线程消费后，路由到队列中，在每个队列一个线程消费。

如何解决数据丢失问题？

答：ack机制、优雅停机

6、E同学会问：消费方执行本地落库的事务成功了，但是ack的时候失败，有重复消费怎么办？

答：在消费方来保证幂等

7、F同学会问：为啥要单独在搞个MQ直接用同步工具不就好了吗？

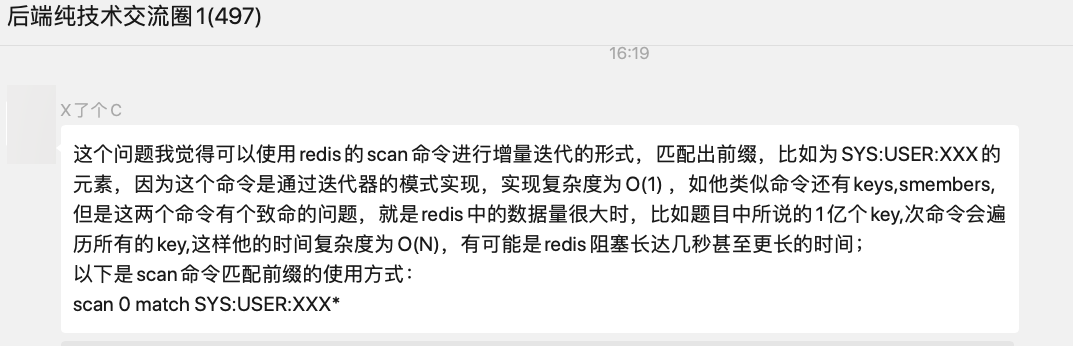
答：比如alibaba otter、离线dataX、kettle等

8、补充说明：整个数据同步过程中，还是需要定期的比对数据；定期的全量同步数据；

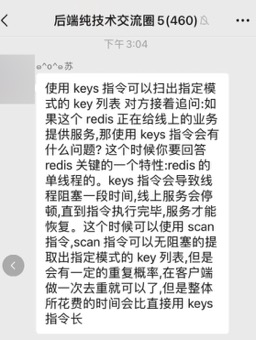
题目三

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并且说明使用的方法的底层原理？

答案一：



答案二:



业务场景四------------

群公告

大家好，上午的思考题不少同学都积极参与讨论，发表自己的观点，这里给一个参考思路供大家参考。欢迎大家后续也积极参与讨论，共同打造一个高质量的技术交流社群。

=========题目回顾：

一、业务场景

小猛最近要将公司的订单表进行分库分表的设计，但是遇到一个问题在分库分表后，订单id咋生成？

问题：各位同学思考实现全局唯一的id有哪些方案？优缺点有哪些？自己公司目前用的那种方案？

==============下面是参考思路

1、方案一：独立数据表自增id

优点：方便简单，谁都会用

缺点：就是单库生成自增id，要是高并发的话，就会有瓶颈的，表数据还需要定期清理，很麻烦

适用场景：并发较低的场景，数据库要做高可用

2、方案二：uuid

优点：每个系统本地生成，不要基于数据库来了

缺点：uuid太长了，作为主键性能太差了，不适合用于主键

适合场景：随机生成个什么文件名了，编号之类的，你可以用uuid，但是作为主键是不能用uuid的。

3、方案三：获取系统当前时间

优点：方便简单

缺点：并发很高的时候，比如一秒并发几千，会有重复的情况，这个是肯定不合适的

适合场景：当前时间跟很多其他的业务字段拼接起来，作为一个id，比如说订单编号：时间戳 + 用户id + 业务含义编码

4、方案四：snowflake算法

优点：是twitter开源的分布式id生成算法，应用广泛，经历过很多大公司的锤炼，高性能，高并发，集群化，可伸缩

缺点：需要考虑时钟回拨等一系列问题

适用场景：中大型公司，有高并发生成唯一ID场景，同时有能力去解决时钟回拨、多机房部署等生产问题

5、其他开源框架：

（1）美团的leaf：https://tech.meituan.com/2019/03/07/open-source-project-leaf.html

（2）百度的uid-generator：<https://github.com/baidu/uid-generator>

业务场景五

MySQL对于千万级的大表如何优化？

==============下面是参考思路

很多同学第一反应是各种切分；这里给出参考思路顺序是:

1、第一优化你的sql和索引

2、第二加缓存，memcached或redis

3、第三以上都做了后，还是慢，就做主从复制或主主复制，读写分离，可以在应用层做，效率高，也可以用三方工具，第三方工具推荐360的atlas，其它的要么效率不高，要么没人维护

4、第四如果以上都做了还是慢，不要想着去做切分，mysql自带分区表，先试试这个，对你的应用是透明的，无需更改代码,但是sql语句是需要针对分区表做优化的，sql条件中要带上分区条件的列，从而使查询定位到少量的分区上，否则就会扫描全部分区，另外分区表还有一些坑，在这里就不多说了

5、第五如果以上都做了，那就先做垂直拆分，其实就是根据你模块的耦合度，将一个大的系统分为多个小的系统，也就是分布式系统

6、第六才是水平切分，针对数据量大的表，这一步最麻烦，最能考验技术水平，要选择一个合理的sharding key，为了有好的查询效率，表结构也要改动，做一定的冗余，应用也要改，sql中尽量带sharding key，将数据定位到限定的表上去查，而不是扫描全部的表