【简历】面向简历

\*\*1、【分库分表：分片键选择、全局ID生成】选择哪个字段作为分片键分库分表、如何通过基因算法保障分片键易用性（比如一个订单号字段同时支持订单号和用户ID查询）？分库分表后为什么选择雪花算法作为ID？如何保障雪花算法在大规模集群下生成不重复？

简历原文

通过订单号和用户信息复合分片算法完成订单数据分库分表，支持订单号和用户查询维度。

参考链接

用户分库分表：<https://www.yuque.com/magestack/12306/pb98neetmww1rr9y>

乘车人分库分表：<https://www.yuque.com/magestack/12306/zhsauz6ksng8wvgf>

订单分库分表：<https://www.yuque.com/magestack/12306/dyr1d4r3me19gg7l>

分布式雪花算法：<https://www.yuque.com/magestack/12306/ciigw9ctq0v90u3w>

分片键选择考虑因素

选择分片键的关键因素：  
1将经常访问的数据放在同一个分片上  
2保证数据的均匀分布在各个分片上，避免出现热点数据集中

3分片键应该与业务关联紧密，避免跨分片查询和跨库事务的复杂性  
4分片键应该不可变，不能随着业务变化而频繁修改。

用户表、乘车人表选择使用 username 作为分片键

基因法与订单表分片键

1. 用户能查看自己的订单
2. 订单号精准查询

基因法：把用户ID的后6位数据冗余到订单号里（参考淘宝）

分片键：按用户ID 后6位分库分表，使用用户ID作为分片键

分布式雪花算法SnowFlake

#### 雪花算法还是UUID？

不选择UUID的原因：UUID相对较长、无序。

雪花算法ID有序自增，保证B+树的新插入节点分布在树右侧，裂变少，重平衡少，性能更高。

Twitter 雪花算法生成64bit的long：

1. 不使用：1bit，最高位是符号位，0 表示正，1 表示负，固定为 0。
2. 时间戳：41bit，毫秒级的时间戳（41 位的长度可以使用 69 年）。
3. 标识位：5bit 数据中心 ID，5bit 工作机器 ID，两个标识位组合起来最多可以支持部署 1024 个节点。
4. 序列号：12bit 递增序列号，表示节点毫秒内生成重复，通过序列号表示唯一，12bit 每毫秒可产生 4096 个 ID。

默认的雪花算法是 64 bit，具体的长度可以自行配置：

* 如果希望运行更久，增加时间戳的位数；
* 如果需要支持更多节点部署，增加标识位长度；
* 如果并发很高，增加序列号位数。

#### 动态分配标识位——1024 节点下 ID 生成唯一

1、服务通过集群部署，其中部分机器标识位一致  
2、业务存在一定的并发量，同一毫秒下的序列号一致

Mybatis-Plus dataCenterId 和 workerId 生成方案，依赖 Mac 地址和进程 PID。虽然能做到尽量不重复，但仍有小几率。：

* dataCenterId 的取值与 Mac 地址有关
* 通过 MAC + PID 的 hashcode 获取 16 个低位，进行运算，最终得到 workerId。

**动态分配：**通过将标识位存放在 Redis、Zookeeper、MySQL 等中间件，在服务启动的时候去请求标识位，请求后标识位更新为下一个可用的。

1. Redis 存储一个 Hash 结构 Key，包含两个键值对：dataCenterId 和 workerId。
2. 在应用启动时，通过 Lua 脚本去 Redis 获取标识位。dataCenterId 和 workerId 的获取与自增在 Lua 脚本中完成，调用返回后就是可用的标示位。
   1. 第一个服务节点在获取时，Redis 可能是没有 snowflake\_work\_id\_key 这个 Hash 的，应该先判断 Hash 是否存在，不存在初始化 Hash，dataCenterId、workerId 初始化为 0。
   2. 如果 Hash 已存在，判断 dataCenterId、workerId 是否等于最大值 31，满足条件初始化 dataCenterId、workerId 设置为 0 返回。
   3. dataCenterId 和 workerId 的排列组合一共是 1024，在进行分配时，先分配 workerId。
   4. 判断 workerId 是否 != 31，条件成立对 workerId 自增，并返回；如果 workerId = 31，自增 dataCenterId 并将 workerId 设置为 0。
3. dataCenterId、workerId 是一直向下推进的，总体形成一个环状。通过 Lua 脚本的原子性，保证 1024 节点下的雪花算法生成不重复。如果标识位等于 1024，则从头开始继续循环推进。

#### 开源分布式 ID 框架——服务节点超出 1024 节点、ID 要求全局唯一

Leaf 和 Uid 都有实现雪花算法，Leaf 额外提供了号段模式生成 ID。  
美团 Leaf：https://github.com/Meituan-Dianping/Leaf  
百度 Uid：<https://github.com/baidu/uid-generator>

雪花算法可以满足大部分场景，如无必要，不建议引入开源方案增加系统复杂度。

\*\*2、【注册：缓存穿透、数据脱敏】在高并发注册场景下，如何解决判断用户名是否已被注册带来的缓存穿透问题？  
用户敏感数据如手机号码、证件号码存储到数据库如何脱敏？

3、封装缓存组件库避免注册用户时，用户名全局唯一带来的缓存穿透问题，减轻数据库访问压力。

3、【登录：多方式读请求扩散】系统支持用户名、手机、邮箱登录，登录时无法确定用户的分片键，造成的“读请求扩散”问题如何解决？

4、【高并发查询】众多条件如何满足？

查询站点对应的列车车次信息。

* 你以为：通过搜索引擎技术ElasticSearc解决，因为涉及大量的查询条件。比如：出发车站、到达车站、出发时间等。
* 实际上：ElasticSearch 的并发能力以及资源占用情况并不适用海量并发查询。查询条件可以通过Redis 缓存，并在内存中组装。

\*\*5、【高并发购票：限流、一致性】如何限流？如何落库？如何保障余票缓存和数据库一致性？

6、通过 Redis Lua 脚本原子特性，完成用户购票令牌分配，通过令牌限流以应对海量用户购票请求。

5、通过 Redis Lua 脚本原子特性，完成用户购票时票数检验、高铁座位分配以及扣减库存等功能。

4、使用 BinLog 配合 RocketMQ 消息队列完成 MySQL 数据库与 Redis 缓存之间的数据最终一致性。

买一张北京南到南京南的车票。

* 你以为：只扣减北京南到南京南单趟的票。
* 实际上：会扣减北京南-济南西，北京南-南京南，济南西-南京南的三趟车票。如果其中有任意条件不满足都不会购买成功。

\*\*6、【超时订单延时取消】如何取消十分钟未支付的订单？如何避免已支付错误取消？

2、通过 RocketMQ 延时消息特性，完成用户购票 10 分钟后未支付情况下取消订单功能。

【实战】面试实战

美团二面

链接：<https://www.bilibili.com/video/BV1Dn4y1X7aD?vd_source=7341e06df79108c355b8df461a20c071>

**背景：**

12306铁路购票系统，选择开源项目练手，模仿12306完成了核心功能。

**五大模块：**

1. 网关模块：JWT令牌校验、请求路由转发
2. 用户模块：登陆注册、增删改查
3. 购票模块（核心、难点）：查票、购票
4. 订单模块
5. 支付模块

**难点和解决：**

主要在于购票模块高并发与可靠性、查票购票一整个流程如何操作

难点1：查询，南京-北京，需要展示南京南-北京x的一系列

解决1：

地区映射表，每个地区对应哪些车站，放入缓存

难点2：购票，参数校验

解决2：

责任链设计模式，非空判断、购票日期小于当前、是否和已有车票冲突

Q：车次出发时间冲突如何解决？

A：bitmap位图，用户一天24h，以10min为单位，一天6\*24=144比特位。

用户购买一张票后，举例说0点到1点，就需要将0-5这6个比特位填充。再次下单时候，查看有无重叠。

Q：每个用户一个bitmap？

A：以10min为单位，一个用户一天18个字节，一亿个用户占用0.7G内存，日活跃用户2-3kw，300MB，Rdis部署了很多集群，为了给用户更高的响应速度，内存消耗可以接受。如果以20min，占用会更小。

Q：如果真有一亿用户，如何快速检索用户对应的bitmap？

A：第二种解决方案，针对车次建立bitmap，看用户下单了哪些车次

Q：对车次bitmap存储，一个用户给全家人买了票，购票数量偏多，20+，需要merge时，这种数据结构的优势？

A：CPU位操作很快，其他点？

Q：现在是01编码，发散开来如果存长文本/复杂文本，比如头条场景/论文查重率，如何解决重复校验问题？

A：哈希操作+布隆过滤器，针对字符串进行去重

Q：这个只能检测一模一样。查重率？

A：对A分词后对关键词存入布隆过滤器，对B再来一次能不能命中。

Q：了解一下瓦片算法，可以用滑动窗口做分组哈希。

难点3：传统秒杀可以Redis库存扣减，12306提前15天放票均匀分布

解决3：

* 对于非节假日情况，数据库+分布式锁，保证同一用户同一时间只下一单；
  + 分布式锁的粒度优化
    - 如果根据train id设置锁key，会导致同一辆车次只会让一个用户操作
    - 优化，多拼接一个座位类型（商务座、一等座、二等座）
    - 优化，a->b->c，a->b和b->c可以并发，受到MySQL间隙锁启发，间隙有重叠再进行阻塞
  + 十万个请求争夺分布式锁 -> Redis的big key问题
    - 微服务有很多JVM实例，比如5个，通过网管模块进行负载均衡，每个实例2w个请求，先竞争本地的Reentrantlock锁，再竞争分布式锁，因此最终只有5个线程竞争分布式锁
* 对于节假日抢票，TOKEN令牌限流，1比1映射票数不保证强一致性，取到令牌才可以购票，没票设置为-1挡住后续无效请求
  + Q：为什么获取TOKEN不需要强一致性？
  + A：Redis中还有余票缓存，这个是保证强一致的。Token令牌是隐式的，用户看不见，是用于限流的，所以保有一定冗余
    - 令牌有、余票无：拿到Token还是会进行兜底查询的
    - 令牌0：查数据库、更新Redis缓存。如果还是没有，设置为-1，打回无效请求
  + Q：余票缓存和数据库如何保持一致？
  + A：canal监听binlog发给MQ消费消息实现。
    - 保证强一致性的情况：购票后删除Redis缓存，会过于频繁而且还是要查数据库
    - 保证最终一致的情况：canal监听binlog，会存在1-2s的延迟，但是可以接受。12306官网并不会因为我下单一张票就立即扣减

**项目独立思考的部分：**

锁粒度降低、车票时间冲突

1.3w行源码->删减增加->1w行源码

**学习思路、认知提升思路，项目经验：**

改良已有项目

理清楚前人代码（单体项目？微服务项目？）

针对具体模块，根据git提交记录了解具体职能，与其他接口的调用关系

交流探讨代码风格相关

**实习遇到问题的解决思路：**

需求难以实现的话，看一下技术栈，跟导师沟通需求紧急程度

参考前人类似处理案例

网络资源学习

**反问：**

Q：实习生会接收到哪些任务？

A：（1）代码测试，辅助提升性能质量。（2）在已有项目迭代需求

Q：我的面试哪里可以改进。美团是我第一个比较正式的面试，其他面试也就问了30min，拿了offer不太想去。

A：对项目熟练度较高，表达比较完善。