【简历】面向简历

1、分库分表

模拟问题

1. 选择哪个字段作为分片键分库分表？
2. 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）？
3. 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. 分库分表后为什么选择雪花算法作为ID？
5. 如何保障雪花算法在大规模集群下生成不重复？

简历描述

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)
* [**手摸手实现分布式ID组件库**](https://www.yuque.com/magestack/12306/lc2yb8gxtvfdt7rp)

分片键选择

在订单分库分表中，我们选择**用户ID**和**订单号**作为复合分片键。用户ID支持用户维度的查询，订单号支持订单维度的查询。为了避免查询时同时携带两个字段，我们采用了**基因算法**。

基因算法

基因算法的核心是将用户ID的后六位冗余到订单号中。这样，订单号不仅包含自身信息，还携带了用户ID的后六位。通过这种方式，我们只需按照用户ID的后六位进行分库分表，即可在查询时通过订单号或用户ID快速定位到具体分片，避免了读扩散问题。

**示例：**

* 用户ID：568576
* 订单号：336951470136448568576

通过基因算法，订单号末尾的568576即为用户ID的后六位，确保了查询的灵活性。

选择雪花算法的原因

1. **高性能**：雪花算法在内存中生成ID，不依赖数据库，生成速度快。
2. **有序性**：生成的ID基于时间戳递增，适合数据库索引，提升查询效率。
3. **高吞吐**：每秒可生成数百万ID，满足高并发场景。

保障ID不重复

1. **标识位分配**：雪花算法通过5bit数据中心ID和5bit工作机器ID（共10bit）标识节点，最多支持1024个节点。通过确保每个节点的标识位唯一，避免ID重复。
2. **动态分配标识位**：在集群环境下，可以通过Redis或Zookeeper动态分配标识位，确保每个节点获取唯一标识，避免冲突。

**示例：**

* 时间戳：41bit（69年）
* 数据中心ID：5bit（32个数据中心）
* 工作机器ID：5bit（32台机器）
* 序列号：12bit（每毫秒4096个ID）

通过合理分配标识位，确保大规模集群下ID的唯一性。

2、注册

模拟问题

1. 高并发注册如何解决判断用户名是否已被注册带来的缓存穿透？
2. 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？

简历描述

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。

参考文档

* [**手摸手之注册用户如何防止缓存穿透？**](https://www.yuque.com/magestack/12306/go6vg8whk9g1lyhp) 
  + [**用户注册布隆过滤器容量设置以及碰撞率问题**](https://www.yuque.com/magestack/12306/fr3zurztaq3xwfkc)
* [**手摸手之实现敏感信息加密存储**](https://www.yuque.com/magestack/12306/vhf4i3c604t2qex7)
  + 核心技术文档-如何防止用户敏感数据泄露 shardingsphere文档
* [**手摸手之用户敏感信息展示脱敏**](https://www.yuque.com/magestack/12306/myl4gqx84bxyxmay)
* [**如何防止用户敏感数据泄露**](https://www.yuque.com/magestack/12306/cd9zbuugg663qsu4)

缓存穿透

在高并发注册场景下，判断用户名是否已被注册时，缓存穿透问题会导致大量请求直接访问数据库，增加数据库负载。我们通过以下方案解决：

1. **布隆过滤器**：
   * 将所有已注册的用户名存入布隆过滤器，用于快速判断用户名是否可能已存在。
   * 如果布隆过滤器判断用户名不存在，则该用户名一定未被注册，直接返回可用，避免查询数据库。
   * 如果布隆过滤器判断用户名可能存在，则进一步查询缓存或数据库。
2. **缓存层**：
   * 使用Redis缓存已注销的用户名，支持用户名复用。
   * 如果布隆过滤器判断用户名可能存在，但缓存中不存在，说明该用户名已被注销，可以被再次使用。
3. **异常行为限制**：
   * 限制用户频繁注销和注册的行为，例如每个证件号最多允许注销5次，防止缓存层膨胀。
4. **缓存分片**：
   * 对Redis缓存进行分片，避免大Key问题，提升查询性能。

**总结**：通过布隆过滤器+缓存层的双重机制，既避免了缓存穿透，又支持用户名复用，同时通过异常行为限制和缓存分片保障了系统的稳定性和性能。

数据脱敏

用户敏感数据（如手机号、证件号）的脱敏存储通过以下方式实现：

1. **数据加密**：
   * 使用ShardingSphere的加密模块，在数据写入数据库时自动加密，读取时自动解密。
   * 加密过程对业务透明，无需修改SQL或业务逻辑。
2. **加密配置**：
   * 在application.yaml中配置ShardingSphere Driver，指定加密规则和算法。
   * 在shardingsphere-config.yaml中定义敏感字段的加密策略。
3. **数据清洗**：
   * 对于历史明文数据，通过不停机清洗任务将其转换为密文，确保数据一致性。
4. **安全性**：
   * 加密算法采用行业标准（如AES），确保数据安全性。
   * 加密密钥通过安全渠道管理，防止泄露。

**总结**：通过ShardingSphere实现敏感数据的透明加密存储，既满足安全合规要求，又减少了对业务逻辑的侵入。

3、查询

模拟问题

1. 众多查询条件如何满足？

参考文档

* [**手摸手之车票搜索为什么用Redis而不是ES？**](https://www.yuque.com/magestack/12306/zd9wok8w0dn8eig5)
* [**手摸手之如何完成列车数据检索**](https://www.yuque.com/magestack/12306/tygc8hs113al2c2z)

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。

4、购票

模拟问题

1. 如何限流？
2. 如何落库？
3. 如何保障余票缓存和数据库一致性？

简历描述

* 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。
* 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。
* 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。

参考文档

* [**节假日高并发购票Redis能扛得住么？**](https://www.yuque.com/magestack/12306/eq6v9p9dfre117mg)
* [**手摸手之实现列车购票流程**](https://www.yuque.com/magestack/12306/nmmgqkgbfxb2bwl0)
  + [**手摸手之实现用户购票责任链验证**](https://www.yuque.com/magestack/12306/ggg2txzbfgfqp6tm)
  + 核心技术文档-从根上理解Redis分布式锁演进架构
* [**手摸手之实现v2版本列车购票流程**](https://www.yuque.com/magestack/12306/ov3u6lpgartx0mst)（Lua脚本不太会）
  + [缓存击穿之双重判定锁如何优化性能？](https://www.yuque.com/magestack/12306/xrtg5mibquardvvi)（逻辑清晰、简明易懂）
  + [如何扣减列车座位库存](https://www.yuque.com/magestack/12306/glv5e0785b2d7oag#L6qWt)
  + [**高并发库存扣减为什么需要令牌限流？**](https://www.yuque.com/magestack/12306/gdqnbhp10xx9g9kv)
* [**购买列车中间站点余票如何更新？**](https://www.yuque.com/magestack/12306/efhhtogdr8ouz6t2)
  + [**缓存与数据库一致性如何解决？**](https://www.yuque.com/magestack/12306/wocbrht50ctg14nv)（加星，很详尽）
  + [**手摸手之列车余票如何保障缓存数据库一致性**](https://www.yuque.com/magestack/12306/glv5e0785b2d7oag)
* [**余票Binlog更新延迟问题如何解决？**](https://www.yuque.com/magestack/12306/kxuug8l6zslfyz21)
  + [**购买列车余票如何防止库存超卖？**](https://www.yuque.com/magestack/12306/pemih6p8h7xaw2b3)

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。

5、超时

模拟问题

1. 如何取消十分钟未支付的订单？
2. 如何避免已支付错误取消？

简历描述

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。

参考文档

* [**订单延时关闭功能技术选型**](https://www.yuque.com/magestack/12306/sazb1bn2a1okwtlu)（最后选择RocketMQ实现）
* [**创建订单并支付后延时关闭订单消息怎么办？**](https://www.yuque.com/magestack/12306/ldw8nxp96yfg7cgx)
  + 手摸手之消息队列正确使用姿势（看不大懂，要补）

6、优化

参考文档

* 缓存击穿之双重判定锁如何优化性能？（逻辑清晰、简明易懂）
* 12306核心接口性能优化都做了什么？（有点难噢）

【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。