【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。

参考文档

* [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)

1. 选择哪个字段作为分片键分库分表？

在订单分库分表的场景中，选择**订单号**和**用户ID**作为复合分片键。这样可以支持通过订单号和用户ID两个维度进行查询，同时避免数据倾斜问题。

2. 如何通过基因算法保障分片键易用性？

基因算法通过将用户ID的哈希值嵌入到订单号中，使得订单号既包含订单的唯一标识，也包含用户ID的信息。这样，在查询时可以通过订单号直接定位到分片，同时也可以通过用户ID快速找到相关的订单数据，避免了全表扫描。

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？

为解决“读请求扩散”问题，可以采用**映射表**的方式。将用户名、手机、邮箱等信息与用户ID建立映射关系，存储在单独的索引表中。登录时先通过索引表查询到用户ID，再根据用户ID定位到具体的分片，从而避免全表扫描。

4. 分库分表后为什么选择雪花算法作为ID？

雪花算法生成的ID具有以下优点：

* **全局唯一**：通过时间戳、机器ID和序列号组合，确保ID在分布式系统中唯一。
* **有序性**：ID按时间递增，适合作为数据库主键，避免索引分裂。
* **高性能**：本地生成ID，无需依赖数据库或外部服务，生成速度快。

5. 如何保障雪花算法在大规模集群下生成不重复？

在大规模集群下，需要确保每个节点的**机器ID**唯一。可以通过以下方式实现：

* **集中分配机器ID**：使用Zookeeper或Redis等分布式协调服务，集中分配和管理机器ID。
* **配置文件分配**：在部署时为每个节点配置唯一的机器ID，确保不重复。
* **动态检测和冲突处理**：在节点启动时检测机器ID是否冲突，若冲突则自动调整。

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](https://www.yuque.com/magestack/12306/cd9zbuugg663qsu4)
* [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)

1. 如何解决判断用户名是否已被注册带来的缓存穿透？

#### 什么是缓存穿透？

请求不存在数据库的数据 -> 数据库负载、缓存内存耗尽、用户体验差

#### 用户注册场景下的缓存穿透？

注册时判断用户名是否已存在

#### 缓存穿透的常见方案？

1. 对不存在的 Key 缓存，值设为Null，并设置短暂过期时间，如 60 秒。  
   缺点：尝试却没注册一个不存在的用户名，该值60s内都不可被注册
2. 使用布隆过滤器存入已注册用户名，如果用户名在布隆过滤器中，再查询缓存或数据库，不在布隆过滤器的一定不存在。  
   缺点：布隆过滤器不能删除元素，注销的用户名无法再次使用。
3. 使用Redis Set存储已注册用户名，检查是否在集合内。  
   缺点：占用内存。
4. 查询缓存不命中时使用分布式锁来保证只有一个线程访问数据库。  
   缺点：注册高峰期其他用户注册请求缓慢或超时

#### 12306如何解决注册穿透？

**布隆过滤器 + Username Redis Set缓存**

1. 布隆过滤器不存在，说明数据库一定没有，可用
2. Username Redis Set缓存存在，说明已注销，可用
3. 判断数据库到底有没有

**潜在问题：**用户频繁申请后注销导致Username Redis Set过大  
Redis大Key问题是指在Redis中存在单个Key对应的Value数据量过大。

**解决方案：**

1. 限制证件号最多注销5次
2. 对缓存进行分片处理，根据用户名的HashCode进行取模操作，将数据分散存储在多个Set结构中。

#### 扩展问答

Q：现在redis中有一个大key，如何平滑的将这个大key拆分？

A：双写：大的redis key正常写入，同时还要写一个拆分后的。在某一个时机两边数据能够对齐的情况下，我们给他从大 Key迁移到新的Key上。

2. 用户敏感数据（如手机号、证件号）存储到数据库如何脱敏？

#### Apache ShardingSphere

开源分布式数据库解决方案

由 JDBC、Proxy、Sidecar（规划中）组成

其中的 JDBC 组件支持数据分片、分布式事务、读写分离、数据加密

配置 shardingsphere-config.yaml：

cipherColumn加密字段

encryptorName加密算法（AES）

原理：拦截、解析、改写、发送

#### 1、数据源配置

加密前，项目中数据源如druid或hikari直接通过Spring加载到IOC 容器

加密后，由 ShardingSphere包装一层再交给 Spring 管理

#### 2、加密器配置

ShardingSphere 中内置了 AES 和 MD5 两种加密算法

#### 3、脱敏表配置

* plainColumn（明文列）：原始密码列
* cipherColumn（密文列）
* logicColumn（逻辑列）：面向开发者编写SQL，和明文以及密文列保持一种映射关系

#### 4、查询属性配置

数据库中同时存储了明文列以及密文列时，该属性决定了

* 是查询明文列的数据直接返回
* 还是查询密文列再通过 ShardingSphere 解密后返回。
* **静态数据脱敏**：适用于将数据从生产环境脱敏后分发至测试、开发等场景。常用的脱敏方法包括替换、无效化、乱序、对称加密等。
* **动态数据脱敏**：在生产环境中实时对敏感数据进行脱敏，根据不同角色和权限执行不同的脱敏方案。
* **无效化和随机值替换**：通过特殊字符（如\*）替换敏感数据，或者使用随机值替换真实数据，以保持原有数据格式的同时保护隐私。
* **对称加密**：对敏感数据进行加密存储，通过密钥解密可以恢复原始数据，确保数据的安全性。

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。

1. 众多查询条件如何满足？

TicketController#pageListTicketQuery

在12306车票搜索场景中，虽然页面上有众多查询条件（如出发地、目的地、出发日期、车次类型、席别等），但这些条件并不会全部触发后端请求。大多数条件是由前端进行筛选的，只有在点击“查询”按钮时才会真正触发后端的请求。

1. 验证数据正确：责任链验证
   1. 空
   2. 判断出发日期不能小于当前日期，毕竟不可能买上一天的车票
   3. 验证出发地和目的地是否存在
   4. 查询出发站点和到达站点是否存在
2. 加载城市数据
   1. 搜索北京南到杭州东，列出北京到杭州所有的列车车次

通过站点关联到城市，通过城市查询列车

1. 查询站点信息
   1. 列车站点数据结构  
      Key：Region\_train\_station\_起始城市\_终点城市\_日期  
      Value：  
       Key：列车ID\_起始站\_终点站  
       Val：列车详细信息
   2. 使用Redis存储列车数据，键的设计采用“出发地\_目的地\_日期”的形式，这样可以快速定位到符合条件的列车数据。
   3. 拼接Key查询Redis，如果为空分布式锁+双重判定锁
   4. 按出发时间排序
2. 查询余票信息  
   列车余票数据是实时变更的，如果在存储到基本信息中，就没办法变更了，所以单独存储
3. 构建返回数据
   1. 12306 列车查询页可知，会存在不同的查询条件，这些查询条件都是通过本次查询所有列车数据构建出来的。页面上的大部分查询条件（如车次类型、席别、出发车站等）由前端进行筛选，减少后端请求的压力。
   2. 采用构建者模式

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](https://www.yuque.com/magestack/12306/ag5pffwexihshe2s)
* [**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）
  + [**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。

1. 如何限流？

根据链接内容，在高并发购票场景中，限流通过 **Redis Lua 脚本的原子特性** 实现令牌限流机制。具体步骤如下：

* **令牌容器**：将未出售的座位作为令牌放入 Redis 容器中，用户购票时需要从容器中获取令牌，成功获取令牌的用户才能进入后续流程，否则直接返回失败。
* **Lua 脚本**：通过 Lua 脚本原子性地检查令牌容器中的令牌数量，并扣减令牌，确保在高并发下不会超卖。
* **分布式锁优化**：为了避免大量用户同时请求分布式锁，仅允许少量用户竞争锁，减少 Redis 的压力。

2. 如何落库？

根据链接内容，在用户购票成功后，订单数据需要落库。具体流程如下：

* **订单创建**：通过分布式锁保障座位分配的唯一性，生成订单号（基因法）并创建订单记录。
* **延迟关闭订单**：通过 RocketMQ 的延时消息特性，实现订单超时未支付自动取消功能。
* **分库分表**：通过订单号和用户信息的复合分片算法，完成订单数据的分库分表，支持订单号和用户维度的查询。

3. 如何保障余票缓存和数据库一致性？

根据链接内容，为了保障余票缓存和数据库的一致性，可以采用以下方案：

* **BinLog + RocketMQ**：通过 BinLog 监听 MySQL 的数据变更，将变更消息发送到 RocketMQ，由消费者更新 Redis 缓存，实现最终一致性。
* **Lua 脚本原子操作**：在购票时，通过 Lua 脚本原子性地完成余票校验、座位分配和库存扣减，确保缓存和数据库的数据一致性。
* **双重校验**：在扣减库存时，先检查缓存中的余票数量，再通过 Lua 脚本原子性地扣减，防止超卖。

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](https://www.yuque.com/magestack/12306/dhm3lr598gtt23od)（看不大懂，要补）

1. 如何取消十分钟未支付的订单？

根据链接内容，通过 **RocketMQ 延时消息特性** 实现订单在十分钟未支付的情况下自动取消。具体步骤如下：

* **发送延时消息**：在订单生成时，向 RocketMQ 发送一条延时消息，设置延时时间为十分钟。
* **消息消费**：十分钟后，RocketMQ 会将消息投递给消费者，消费者接收到消息后检查订单状态。
* **取消订单**：如果订单状态仍为未支付，则执行取消订单的逻辑，释放库存并将订单状态标记为已取消。

2. 如何避免已支付错误取消？

根据链接内容，为了避免已支付的订单被错误取消，可以在消费者处理延时消息时进行以下操作：

* **状态校验**：消费者接收到延时消息后，首先检查订单的支付状态。如果订单已支付，则直接忽略消息，不执行取消操作。

6、优化

参考文档

* [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）
* [12306核心接口性能优化都做了什么？](https://www.yuque.com/magestack/12306/bh7c4x3i4sn682bn)（有点难噢）

缓存击穿之双重判定锁优化性能

1. **缓存击穿定义**：热点数据在缓存失效的瞬间，大量并发请求直接访问数据库，导致数据库压力骤增，影响系统稳定性。
2. **双重判定锁机制**：
   * 第一次检查缓存，若数据不存在，获取分布式锁。
   * 获取锁后再次检查缓存，若数据仍不存在，才访问数据库并更新缓存。
3. **优点**：有效减少数据库并发访问，防止缓存击穿，提升系统性能。
4. **Lua脚本实现**：通过Redis Lua脚本保证原子性操作，避免锁竞争和数据不一致问题。
5. **适用场景**：高并发场景下，保障热点数据的缓存和数据库一致性。

12306核心接口性能优化

1. **全面使用缓存**：用NoSQL数据库取代传统数据库，车票查询性能从1000次/s提升至20000次/s，响应时间从1秒降至10毫秒。
2. **队列削峰**：构建交易处理排队系统，接收用户下单请求并异步处理，峰值处理能力超过10万笔/秒。
3. **分库分表**：将订单/电子客票数据从1个节点拆分为3个节点30个库30张表，提升处理能力并支持横向扩展。
4. **读写分离**：用NoSQL数据库存储订单数据，查询性能从200次/s提升至5000次/s，避免高并发查询影响交易处理。
5. **异地双活架构**：构建双活数据中心，支持高峰期弹性扩容，查询能力达30万次/s，确保系统高可用性和可靠性。

【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。