分库分表

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。

* [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？

订单分库分表选择**订单号**和**用户ID**作为复合分片键。支持通过订单号和用户ID两个维度进行查询。

2. 基因算法保障分片键易用性：订单号字段同时支持订单号和用户ID查询

基因算法通过将用户ID的哈希值嵌入到订单号中，使订单号既包含订单的唯一标识，也包含用户ID的信息。

3. 系统支持用户名/手机/邮箱登录，无法确定分片键造成“读请求扩散”？

采用映射表将用户名/手机/邮箱与用户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？

* **全局唯一**：时间戳 + 机器ID + 序列号
* **有序性**：ID按时间递增
* **高性能**：本地生成、速度快，无需依赖数据库或外部服务

5. 如何保障雪花算法在大规模集群下生成不重复？

确保每个节点的**机器ID**唯一：

* **集中分配**：使用Zookeeper或Redis集中分配和管理机器ID。
* **配置文件**：在部署时为每个节点配置唯一的机器ID。
* **启动检测**：在节点启动时检测机器ID是否冲突。

用户注册

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。

* [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？

缓存穿透

请求不存在于数据库的数据 造成 数据库负载、缓存内存耗尽、用户体验差

缓存穿透的常见方案

1. 对不存在的 Key 缓存并把值设为Null，设置短暂过期时间（如 60 秒）。  
   **缺点：**尝试但没注册一个不存在的用户名，该值60s内都不可被注册
2. 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：**其他用户注册请求缓慢或超时
3. 布隆过滤器存已注册用户名，不在布隆过滤器的一定不存在-可用，如果用户名在布隆过滤器中再查询缓存或数据库，。  
   **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。
4. Redis Set存已注册用户名，检查是否在集合内。  
   **缺点：**占用内存。

12306解决注册穿透：布隆过滤器 + Redis Set缓存

1. 布隆过滤器不存在，说明数据库没有-可用
2. Redis Set缓存存在，说明已注销-可用
3. 查询数据库到底有没有

#### Q：用户频繁申请后注销导致Redis Set。

Redis大Key问题是指在Redis中存在单个Key对应的Value数据量过大。

**A：**

1. 限制1个证件号最多注销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。

#### Q：现在redis中有一个大key，如何平滑拆分？

**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，从大 Key迁移到新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是一个开源分布式数据库解决方案，其中的JDBC支持数据分片、分布式、读写分离、数据加密

1. 数据源配置
   1. 加密前，数据源（如druid或hikari）直接通过Spring加载到IOC 容器
   2. 加密后，由 ShardingSphere包装一层再交给 Spring 管理
   3. 原理：拦截、解析、改写、发送
2. 加密器配置
   1. ShardingSphere 中内置了 AES 和 MD5 两种加密算法
3. 脱敏表配置
   1. plainColumn明文列
   2. cipherColumn密文列
   3. logicColumn逻辑列：面向开发者，和明文以及密文保持映射关系
4. 查询属性配置
   1. 是查询明文列的数据直接返回
   2. 还是查询密文列再通过 ShardingSphere 解密后返回。

车票查询

* [**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)

众多查询条件如何满足？

TicketController#pageListTicketQuery

1. 责任链验证数据
   1. 空
   2. 出发日期≥当前日期，毕竟不可能买上一天的车票
   3. 出发地和目的地存在、出发站点和到达站点存在
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
   2. 通过站点关联到城市，通过城市查询列车
3. 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
4. 单独存储余票信息  
   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
5. 构建返回数据  
   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）由前端进行筛选采用构建者模式构建

车票购买

* 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。
* [**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实际上会扣减北京南-南京南，北京南-济南西，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？

通过**Redis Lua 脚本的原子特性**实现**令牌限流机制**：

* **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**：原子性地检查令牌容器中的令牌数量并扣减，确保不超卖。
* **分布式锁优化**：仅允许少量用户竞争锁，减少 Redis 的压力。

2. 如何落库？

* **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**：RocketMQ的延时消息，实现订单超时未支付自动取消。
* **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？

* **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。

支付超时

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。

* [**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？

**RocketMQ延时消息**：

* 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。
* 十分钟后，RocketMQ 将消息投递给消费者，检查订单状态。
* 如果订单状态仍为未支付，则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？

在消费者处理延时消息时进行**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。

12306优化

* [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**：热点数据缓存失效的瞬间，大量并发请求直接访问数据库，导致数据库压力骤增。
2. **双重判定锁**：

* 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。

1. **Redis Lua脚本：**保证原子性操作，避免锁竞争和数据不一致问题。

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
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
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，避免高并发查询影响交易。
5. **异地双活架构**：构建双活数据中心，支持高峰期弹性扩容。

面试实战

美团二面

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

背景

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：查票、购票
4. 订单模块
5. 支付模块

难点：查询，南京-北京，需要展示南京南-北京x的一系列

解决：地区映射表，每个地区对应哪些车站，放入缓存

难点：购票时的参数校验

解决：责任链设计模式

1. 非空判断
2. 购票日期小于当前
3. 是否和已有车票冲突

Q：车票时间冲突如何解决？

A：针对用户建立bitmap，用户一天24h，以10min为单位， 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。以10min为单位的情况下：

* 一个用户一天144bit = 18B
* 日活跃用户2kw = 2 x 104
* 相乘得到300+ MB

Redis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。

#### Q：如果有一亿用户，如何快速检索用户对应的bitmap？

A：针对车次建立bitmap，看用户下单了哪些车次

#### Q：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：瓦片算法通过滑动窗口分组哈希将长文本转换为哈希集合，结合Jaccard相似度计算，核心优势：

1. ​抗局部修改：重叠窗口设计可容忍部分内容变动。
2. ​高效性：哈希化与MinHash压缩大幅减少数据量。
3. ​可扩展性：适合分布式处理，应对海量文本查重。

难点：传统秒杀可以Redis库存扣减，12306提前15天放票均匀分布

解决：

* 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

项目独立思考部分

锁粒度降低

车票时间冲突

1.3w行源码

认知提升思路

* 改良已有项目
* 理清楚已有代码
* 针对具体模块，根据git提交记录，了解具体职能和与调用关系
* 交流探讨代码风格相关

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。