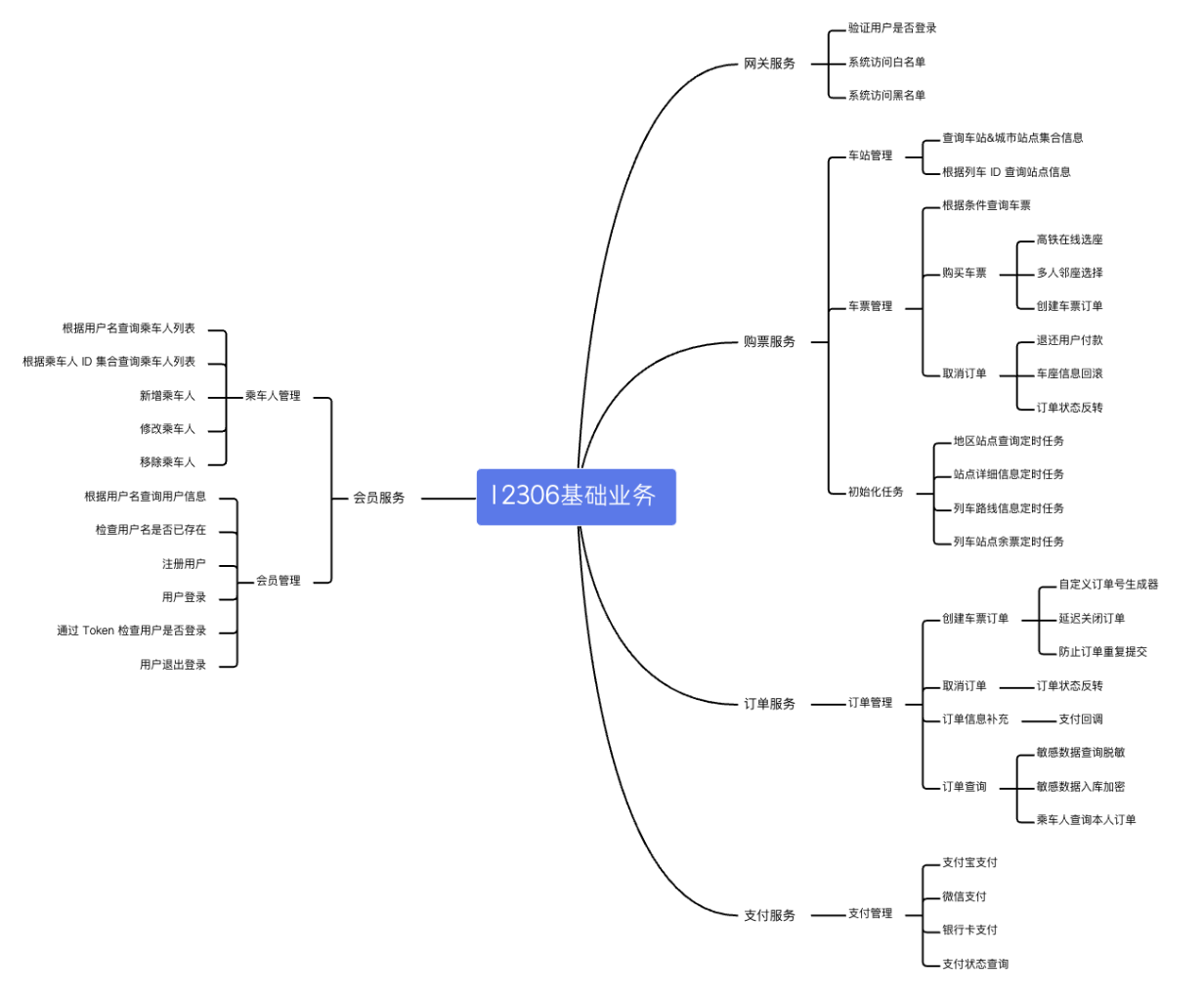
12306 铁路购票系统

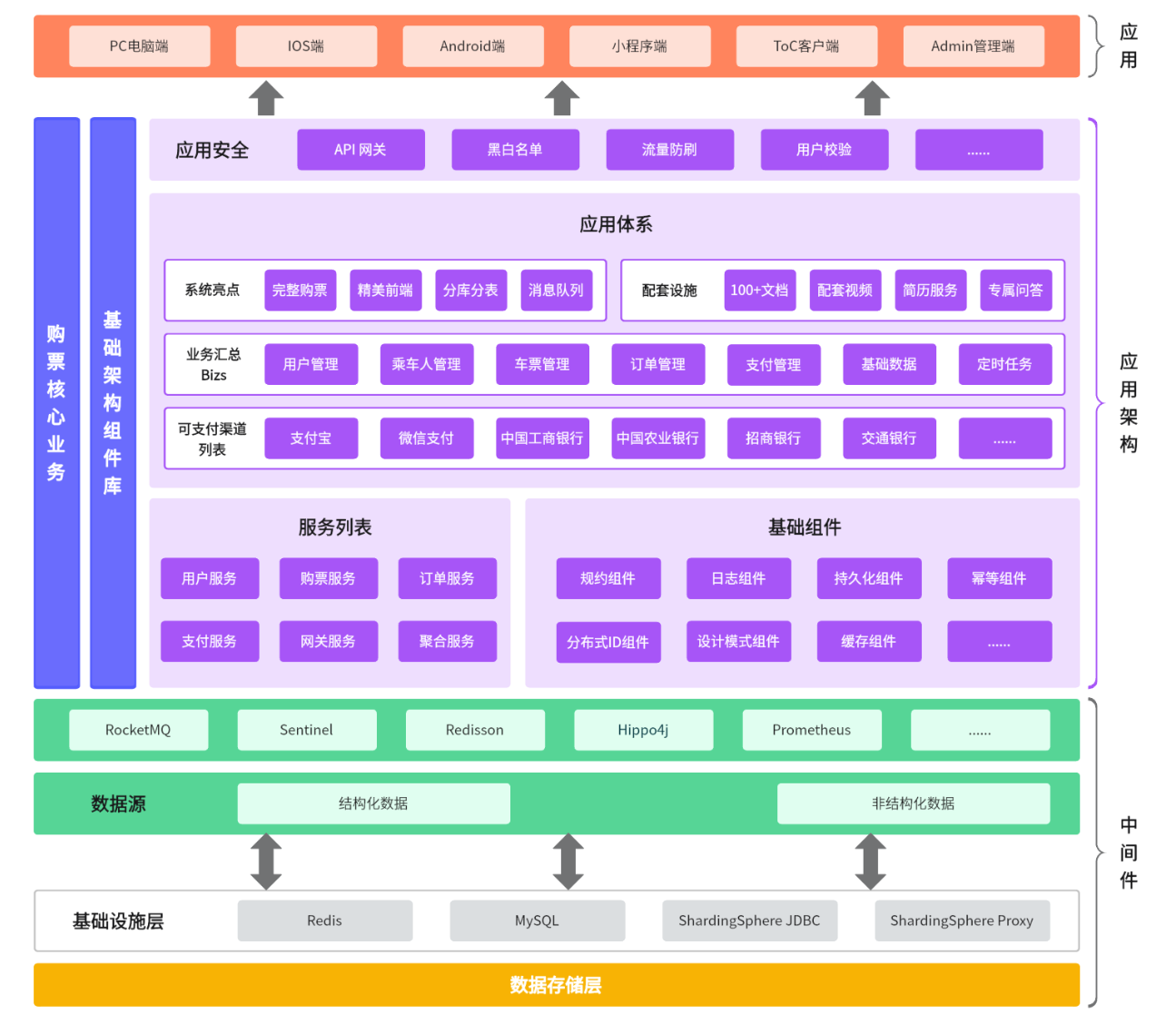
网站文档 links

* [教程文档 语雀 dxxn](https://www.yuque.com/magestack/12306)
* [12306对外文档](https://nageoffer.com/12306/)
* [12306 语雀密码更新](https://wx.zsxq.com/group/51121244585524/topic/1522484552155512)
* <拿个offer网站>
* [星球专栏](https://wx.zsxq.com/group/51121244585524/topic/1522484552155512)
* [接口文档](https://apifox.com/apidoc/shared-86d9d3be-8a15-453f-85a6-4420180a915e)
* [安装文档](https://www.apifox.cn/help/app/web/browser-extension)
* [云服务器Nacos](http://common-nacos-dev.magestack.cn:8848/nacos/index.html#/serviceManagement?dataId=&group=&appName=&namespace=&pageSize=&pageNo=)
* [云服务器RocketMQ](http://common-rocketmq-dev.magestack.cn:8088/#/cluster)

项目介绍



采用最新 JDK17 + SpringBoot3&SpringCloud 微服务架构，构建高并发、大数据量下仍然能提供高效可靠的 12306 购票服务。



以用户服务系统为例，低并发和低数据量的系统相对简单，但高并发和海量数据的系统则需要考虑很多额外因素。

1. 当用户在 12306 网站注册新账号或添加乘车人时，系统需验证用户提交信息的真实性和准确性。如何有效预防用户提交虚假信息，保障系统购票的安全？
2. 12306 的大规模用户和乘车人数据如何选择分库分表？选择哪个字段作为分片键？如何在老业务上平滑上线分库分表？出现问题如何快速回滚？
3. 系统支持会员使用用户名、手机号以及邮箱等多种方式进行登录。由于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. 在高并发的会员注册场景下，绝对会出现缓存穿透问题。网上鼓吹的对不存在 Key 进行缓存值设为 Null，以及布隆过滤器等都存在漏洞，如何解决？
5. 存在较多的敏感信息，比如会员或者乘车人的姓名、手机号、邮箱、证件号码以及住址，如何防止数据库被攻击时造成的敏感信息泄露？

再以购票服务为例，当用户购买两个乘车人的高铁一等座票且没有选座时，座位的分配逻辑如下：

1. 首先检查当前列车的一等座余票是否足够。如果余票不足，直接向客户端返回购票请求失败的响应。
2. 获取所有车厢中有两个座位余票的车厢，并对这些车厢进行遍历，按照下述流程执行。
3. 首先检查所有车厢中是否存在一等座车票的相邻座位。如果所有车厢中都没有相邻座位，进入下一步逻辑。
4. 接着检查是否有车厢中包含两个不相邻的一等座座位？因为同车厢两座位相邻座位没有的话，就退而找同车厢不相邻座位。
5. 如果以上逻辑都无法满足，那么最后选择分配不同车厢的不相邻座位。这种情况下，由于已经确认一等座的余票充足，因此一定能够成功完成购票。

背景：假设，有一站列车，途径北京南、济南西、南京南、杭州东。

查询站点对应的列车车次信息。

* 你以为：通过搜索引擎技术 ElasticSearch 技术解决，因为涉及大量的查询条件。比如：车次、车组、出发车站、到达车站、出发时间等。
* 实际上 ：当海量并发查询时，ElasticSearch 的并发能力以及资源占用情况来说，并不适用。而且，大家如果仔细思考，发现这些查询条件都是可以通过类似于 Redis 的缓存技术存储，并在内存中进行组装。

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。

买一张济南西到南京南的车票。

* 你以为：按照上述的逻辑，我如果通过软件恶意刷票，只买济南西-南京南的票，是不是北京南-杭州东就买不到了？
* 实际上：每个站数之间的数量都有规则。虽然放票时间都是一致的，但是优先大站之间的票量，避免因为大量用户购买了中间站的车票导致始发站和终点站的购票困难。该问题通过动态放票解决，比如刚开始放票时对小站之间仅开放少量票，大站之间放出来多数票。如果后续接近发车时间，再开放小站间的车票。

如何判断一份学生的简历含金量怎么样？通常由以下项进行排序：

学校&学历 > 获奖经历 > 实习和工作经验 > 项目 > 证书 > 专业技能 > ......

[如何学习12306](https://nageoffer.com/12306/how-to-study/)

12306 铁路购票系统学习总体分为三块：

1. 组件库开发
2. 业务梳理
3. 业务系统开发

组件库开发

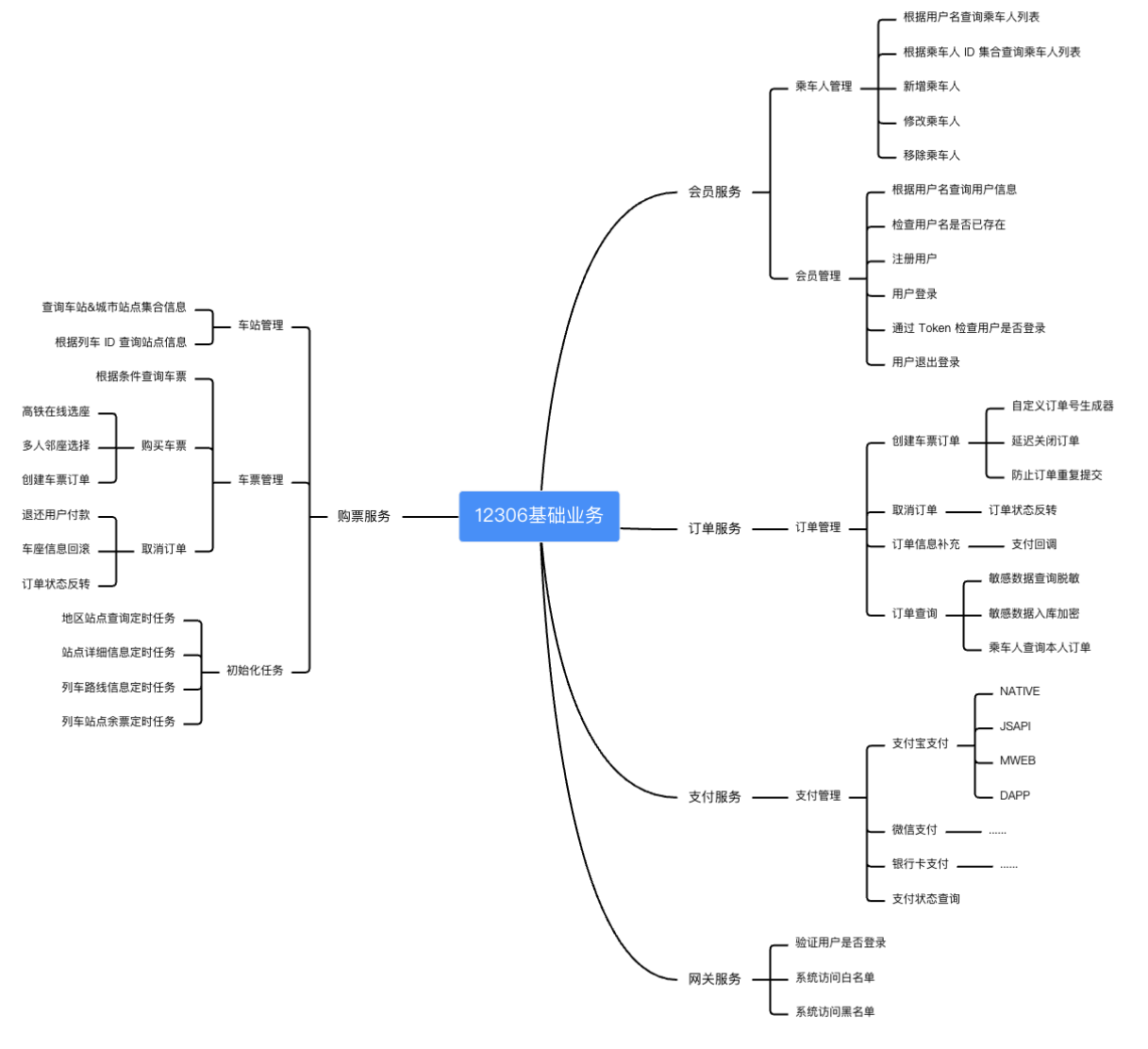
组件库的产出源于对公共功能的封装，避免了在不同项目之间相互复制代码的情况。

目前已有组件如下，可能新增加的组件更新不及时，实际以代码库 /frameworks 目录下为准。



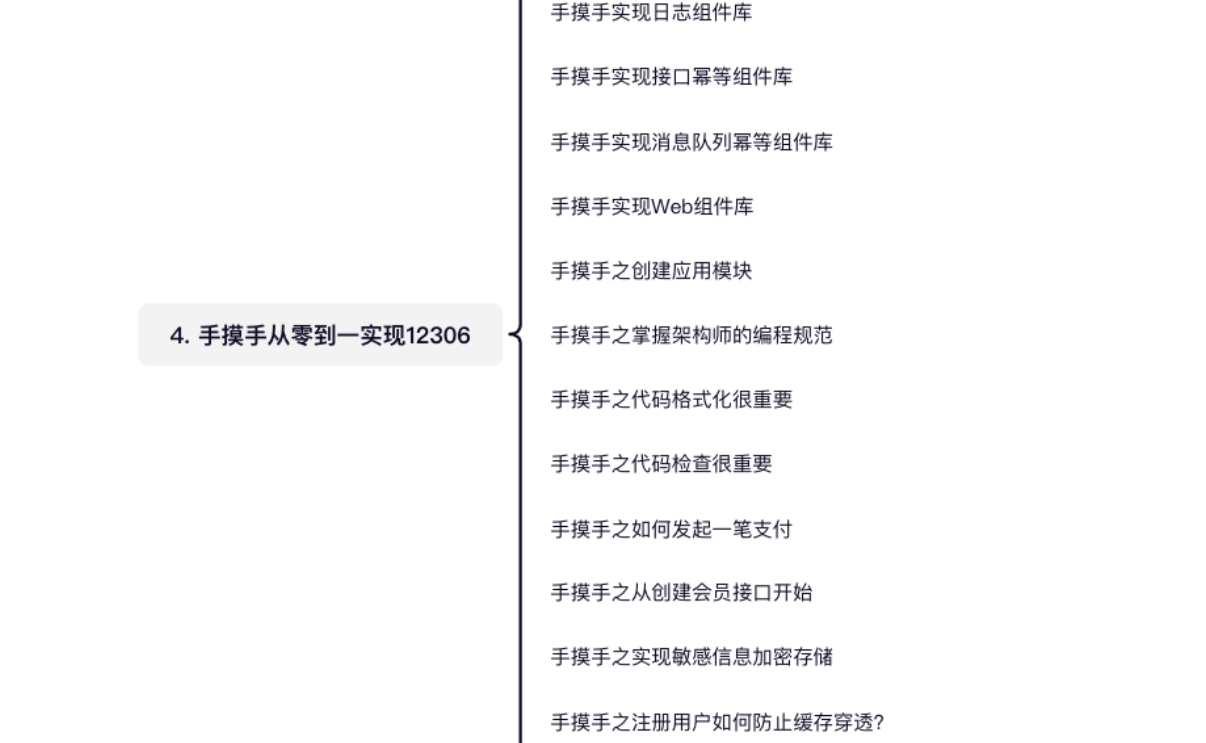
业务梳理

在 12306 铁路购票系统中，包括会员、购票、订单、支付以及网关服务。



系统开发

我们在手摸手从零到一开发章节中，会有非常详细的系列教程，帮助大家梳理以及开发。



云服务器

经常听到星球里小伙伴和我吐槽，安装中间件环境太麻烦了，总会遇到各种错误，严重影响个人开发进度。

为了解决这个问题，我特地购买了两台云服务器，搭建了各种中间件环境，比如：RocketMQ、Redis、Nacos、Prometheus、Grafana 等，供大家日常使用。

星球视频

星球视频 links

* 星球视频-[12306项目应该怎么跟着星球学习](https://t.zsxq.com/19vdYME6k)
* 星球视频-[12306项目快速启动教程](https://t.zsxq.com/199eo7PNQ)
* 星球视频-[如何发起一笔购票订单支付流程](https://t.zsxq.com/19m8jCsEu)
* 星球视频-[购买车票出现站点余票不足如何解决](https://t.zsxq.com/19r6N7Xua)
* 星球视频-[12306 项目不同模块都是做什么的](https://t.zsxq.com/19GyTsbcN)
* 星球视频-[手摸手梳理数据库表关系之用户管理](https://t.zsxq.com/190bPtXyA)
* 星球视频-[手摸手梳理数据库表关系之订单管理](https://t.zsxq.com/19jdoC6WP)
* 星球视频-[手摸手梳理数据库表关系之列车数据管理](https://t.zsxq.com/19XzP4tkx)
* 星球视频-[如何实现列车购票责任链验证请求数据准确性](https://t.zsxq.com/19i17157c)

星球视频-[如何实现订单延时关闭功能](https://t.zsxq.com/19qbSmiSt)

星球视频-[12306项目应该怎么跟着星球学习](https://t.zsxq.com/19vdYME6k)

**快速启动**

**-> 面试冲刺**

**-> 了解系统（教学视频系列）、针对性学习（手摸手系列）**

星球视频-[12306项目快速启动教程](https://t.zsxq.com/199eo7PNQ)

克隆项目、配置JDK17

初始化数据库、数据表、数据

配置中间件VM参数

记得修改参数：

1、改为自己的名称，防止大家串行；

2、按星球给的公用域名修改redis、rocketmq、nacos

3、Dframework.cache.redis.prefix=windscape0326:

组件库公用云服务器：<https://wx.zsxq.com/group/51121244585524/topic/211854522844251>

-Dspring.data.redis.password=Sm9sVXBOYJjI030b5tz0trjpzvZzRhtZmEbv0uOImcD1wEDOPfeaqNU4PxHob/Wp

-Dspring.data.redis.port=19389

-Dunique-name=windscape0326

-Dframework.cache.redis.prefix=windscape0326:

-Dspring.data.redis.host=common-redis-dev.magestack.cn

-Drocketmq.name-server=common-rocketmq-dev.magestack.cn:9876

-Dspring.cloud.nacos.discovery.server-addr=common-nacos-dev.magestack.cn:8848

启动aggregation-service和gateway-service

访问[nacos控制台](http://common-nacos-dev.magestack.cn:8848/nacos/index.html)验证

用户名/密码：nacos/nacos



启动前端

安装nodejs

问题√：yarn install时报错ETIMEDOUT

描述：There appears to be trouble with your network connection. Retrying...

Error: connect ETIMEDOUT

解决：按照批注建议做了以下操作：

1、把yarn.lock文件删除

2、关闭代理梯子

3、切换yarn镜像源：npm config set registry http://registry.npm.taobao.org

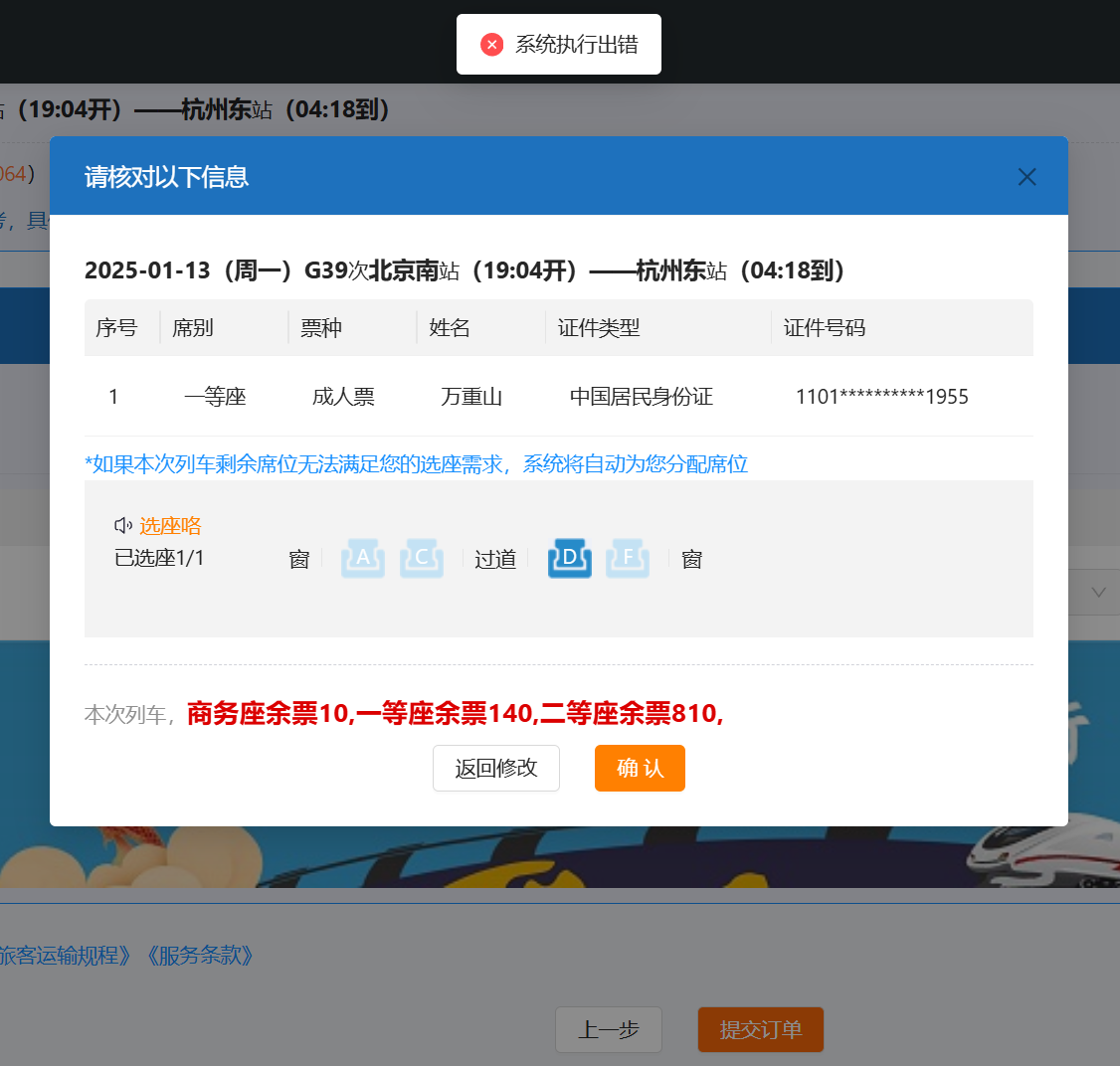
4、重新执行

npm install yarn

yarn install

**success**





问题√：重启IDEA，重新运行AggregationServiceApplication报错

解决：在IDEA中连接本地MySQL的12306数据库后无报错

运行流程：

1、运行AggregationServiceApplication、GateWayServiceApplication

2、右击console-vue后Open In Terminal，输入yarn serve

问题√：提交订单时报错，前端提示系统执行出错，后端报错org.springframework.dao.InvalidDataAccessApiUsageException

解决：vm 参数里有个 cache prefix 参数，最后参数值要有个英文冒号结尾:

参考：https://t.zsxq.com/lr0Xq

其他解决1：可以看一下rocketmq的broker和server启动了没

其他解决2：基本问题是在mq, 看看云服务器端口有没有开10909和10911

星球视频-<购买车票出现站点余票不足如何解决>

场景：商务座剩余1，但需要购买2

座位的三种状态：可售、锁定（已下单待支付）、已售

Key：商务座（0），一等座（1），二等座（2）

Reset接口可以重置座位数目

问题：延长票怎么实现

描述：被问到延长票怎么实现……回答说是正常的再次购票，面试官说好像不对。所以延长票逻辑该怎么设计？

星球视频-[12306 项目不同模块都是做什么的](https://t.zsxq.com/19GyTsbcN)

语雀文档-[**手摸手之工程目录结构如何设计**](https://www.yuque.com/magestack/12306/bvypcmc38cdvgt56)

星球视频-[手摸手梳理数据库表关系之用户管理](https://t.zsxq.com/190bPtXyA)

语雀文档-[手摸手之梳理数据库表关系](https://www.yuque.com/magestack/12306/ihi1rsmxsl5fq0u4)

Q：为什么需要单独的邮箱表和手机号表？

A：

* 分库分表分片键用的是username，ShardingSphere 会通过分片键 username 用户名来确定数据在哪个库中的哪个表。所以，分库分表后，每次增删改查都需要带上分片键用户名。不然的话，查询会请求所有库的所有用户表，新增、修改和删除会直接报错。
* 实际业务需求中，是可能不输入用户名，而是通过“邮箱+密码”“手机+密码”的方式登录的。
* 因此单独的邮箱表和手机号表帮助查找到分片键 username。

星球视频-[手摸手梳理数据库表关系之订单管理](https://t.zsxq.com/19jdoC6WP)

语雀文档-[手摸手之梳理数据库表关系](https://www.yuque.com/magestack/12306/ihi1rsmxsl5fq0u4)

t\_order\_item\_passenger：用于代人购票时，被代的乘车人还未注册（即没有用户名），但后来注册了，可以通过购票时的身份证号关联订单。

星球视频-[手摸手梳理数据库表关系之列车数据管理](https://t.zsxq.com/19XzP4tkx)

语雀文档-[手摸手之梳理数据库表关系](https://www.yuque.com/magestack/12306/ihi1rsmxsl5fq0u4)

Q：列车站点关联表没听懂是用来干什么的

A：假如你要买北京南到沧州西的车票，需要同时扣除北京南->天津西，北京南->沧州西，天津西->沧州西的车票余量，如果没有这个关联表，你需要在内存中进行重复计算，性能一般，所以我们通过提前缓存的方式解决该问题。

Q：那这个查余票是在哪个表查询呢

A：余票在缓存里

星球视频-[如何实现列车购票责任链验证请求数据准确性](https://t.zsxq.com/19i17157c)

语雀文档-[手摸手之实现用户购票责任链验证](https://www.yuque.com/magestack/12306/ggg2txzbfgfqp6tm)

星球视频-[如何实现订单延时关闭功能](https://t.zsxq.com/19qbSmiSt)

语雀文档-[订单延时关闭功能技术选型](https://www.yuque.com/magestack/12306/sazb1bn2a1okwtlu)

语雀文档

⭐[核心业务&技术](https://www.yuque.com/magestack/12306/un0ded7hs43s7o7c#eSz7E)

●判断用户名是否已被注册接口，如何应对海量并发请求带来的缓存穿透问题？  
●用户敏感数据存储到数据库后，如何避免被黑客将数据库攻破并窃取用户信息？  
●海量并发查询车票列表接口，如何应对海量请求的查询，以及众多条件如何满足？  
●海量并发下单接口，如何应对海量用户下单请求，如何进行限流以保证系统不崩溃？以及如何正确落库？  
●用户下单后如何保障列车座位余票缓存和数据库一致性？非常典型的数据库和缓存一致性问题。  
●用户下单十分钟未支付，如何取消用户未付款订单？如果用户已支付，如何避免错误取消？  
●系统中用了消息队列后，如何确保消息不会被重复消费，保证业务系统消费幂等性？  
●海量用户和订单数据如何存储，通过什么分库分表规则保障系统高性能响应用户查询等请求？  
●如何通过分库分表基因算法保障分片键的易用性？比如一个订单号字段同时支持订单号和用户 ID 查询。  
●海量数据分库分表后，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 个人收获

1、使用责任链模式重构请求数据准确性检验，比如：查询购票、购买车票下单以及支付结果回调等业务。

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。

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。

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。

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。

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。

7、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。

8、创建订单明细与乘车人的关联表，分库分表规则同订单，完成乘车人账号登录查询本人车票功能。

[中间件学习 RocketMQ](https://nageoffer.com/docs/rocketmq/)

RocketMQ 分布式消息系统 4.x 版本

什么场景下用 RocketMQ？

**异步解耦**

用户注册后，需要发送注册邮件和短信通知，以告知用户注册成功。

数据流动如下所述：

1. 用户在注册页面填写账号和密码并提交注册信息，这些注册信息首先会被写入注册系统。
2. 注册信息写入注册系统成功后，再同时发送请求至邮件和短信通知系统。邮件和短信通知系统收到请求后分别向用户发送邮件和短信通知。

**流量削峰填谷**

秒杀处理流程如下所述：

1. 用户发起海量秒杀请求到秒杀业务处理系统。
2. 秒杀处理系统按照秒杀处理逻辑将满足秒杀条件的请求发送 RocketMQ。
3. 下游的通知系统订阅 RocketMQ 的秒杀相关消息，再将秒杀成功的消息发送到相应用户。
4. 用户收到秒杀成功的通知。

**顺序消息**

**分区顺序消息**适用于性能要求高，以 Sharding Key 作为分区字段，在同一个区块中严格地按照先进先出（FIFO）原则进行消息发布和消费的场景。

示例

* 用户注册需要发送验证码，以用户 ID 作为 Sharding Key，那么同一个用户发送的消息都会按照发布的先后顺序来消费。
* 电商的订单创建，以订单 ID 作为 Sharding Key，那么同一个订单相关的创建订单消息、订单支付消息、订单退款消息、订单物流消息都会按照发布的先后顺序来消费。

**全局顺序消息**实际上是一种特殊的分区顺序消息，即 Topic 中只有一个分区

**分布式模式缓存同步**

提供一种广播机制，一条消息本来只可以被集群的一台机器消费，如果使用 RocketMQ 的广播消费模式，那么这条消息会被所有节点消费一次，相当于把价格信息同步到需要的每台机器上，取代缓存的作用。

**分布式定时/延时调度**

RocketMQ 提供精确度到秒级的分布式定时消息能力（5.0架构后），可广泛应用于订单超时中心处理、分布式延时调度系统等场景。

**优势：**

* 定时精度高、开发门槛低：消息定时时间不存在阶梯间隔，可以轻松实现任意精度事件触发，无需业务去重。
* 高性能、可扩展：传统的定时实现方案较为复杂，需要进行数据库扫描，容易遇到性能瓶颈的问题，RocketMQ 可以基于定时消息特性完成事件驱动，实现百万级消息 TPS 能力。

RocketMQ 基础概念

**主题 Topic**

消息存储、传输的顶层容器

负责数据的分类隔离、身份和权限

**队列 Queue**

消息存储、传输的实际容器、消息的最小存储单元

一个主题拥有多个队列

**消息 Message**

最小数据传输单元

**生产者 Producer**

通过负载均衡模块选择相应的 Broker 集群队列进行消息投递

**消费者 Consumer**

* 支持以推（push），拉（pull）两种模式对消息进行消费。
* 同时也支持集群方式和广播方式的消费。
* 提供实时消息订阅机制，可以满足大多数用户的需求。

**名字服务器 NameServer**

支持 Topic、Broker 的动态注册与发现。

* Broker管理：NameServer 接受 Broker 集群的注册信息并且保存下来作为路由信息的基本数据。然后提供心跳检测机制，检查 Broker 是否还存活；
* 路由信息管理：每个 NameServer 将保存关于 Broker 集群的整个路由信息和用于客户端查询的队列信息。Producer 和 Consumer 通过 NameServer 就可以知道整个 Broker 集群的路由信息，从而进行消息的投递和消费。

NameServer 通常会有多个实例部署，各实例间相互不进行信息通讯。Broker 是向每一台 NameServer 注册自己的路由信息，所以每一个 NameServer 实例上面都保存一份完整的路由信息。当某个 NameServer 因某种原因下线了，客户端仍然可以向其它 NameServer 获取路由信息。

**代理服务器 Broker**

负责消息的存储/投递/查询、服务高可用保证

Master 与 Slave 的对应关系通过指定相同的 BrokerName、不同的 BrokerId 来定义

BrokerId 为 0 表示 Master，非 0 表示 Slave。

**部署模型小结**

* 每个 **Broker** 与 **NameServer** 集群中的所有节点建立长连接，定时注册 Topic 信息到所有 NameServer。
* **Producer** 与 **NameServer** 集群中的其中一个节点建立长连接，定期从 NameServer 获取 Topic 路由信息，并向提供 Topic 服务的 Master 建立长连接，且定时向 Master 发送心跳。Producer 完全无状态。
* **Consumer** 与 **NameServer** 集群中的其中一个节点建立长连接，定期从 NameServer 获取 Topic 路由信息，并向提供 Topic 服务的 Master、Slave 建立长连接，且定时向 Master、Slave 发送心跳。Consumer 既可以从 Master 订阅消息，也可以从 Slave 订阅消息。

RocketMQ 工作原理

**1. 启动 NameServer。**

相当于一个路由控制中心，等待 Broker、Producer、Consumer 连接

**2. 启动 Broker。**

与所有 NameServer 保持长连接，定时发送心跳包

**3. 创建 Topic**

**4. 生产者发送消息。**

启动时先跟 NameServer 集群中的其中一台建立长连接，并从 NameServer 中获取当前发送的 Topic 存在于哪些 Broker 上，轮询从队列列表中选择一个队列，然后与队列所在的 Broker 建立长连接从而向 Broker发消息

**5. 消费者接受消息。**

跟其中一台 NameServer 建立长连接，获取当前订阅 Topic 存在哪些 Broker 上，然后直接跟 Broker 建立连接通道，然后开始消费消息

动手发一条消息

纯文本和代码有些困难，准备看黑马

RocketMQ 部署架构

本地单副本，生产都是多。

**1. 本地部署-单节点单副本模式**  
因为 Broker 只有一个节点，一旦Broker重启或者宕机时，会导致整个服务不可用。

**2. 本地部署-多节点（集群）单副本模式**  
一个集群内全部部署 Master 角色，不部署 Slave 副本。  
单台机器宕机期间，这台机器上未被消费的消息在机器恢复之前不可订阅，消息实时性会受到影响。

**3. 生产部署-多节点（集群）多副本模式-异步复制**每个 Master 配置一个 Slave，有多组 Master-Slave，HA 采用异步复制方式，主备有短暂消息延迟（毫秒级）

Master 宕机，磁盘损坏情况下会丢失少量消息。

**4. 生产部署-多节点（集群）多副本模式-同步双写**

有多对 Master-Slave，HA 采用同步双写方式，即只有主备都写成功，才向应用返回成功。  
性能比异步复制模式略低（大约低10%左右），发送单个消息的RT会略高，且目前版本在主节点宕机后，备机不能自动切换为主机。

[中间件学习 Sharding-JDBC](https://nageoffer.com/docs/shardingsphere-jdbc/)

分布式的数据库生态系统，可以将任意数据库转换为分布式数据库，并通过数据分片、弹性伸缩、加密等能力对原有数据库进行增强。

ShardingSphere 5.3.2 版本

分库分表背景

传统的将数据集中存储至单一节点的解决方案，在性能、可用性和运维成本这三方面已经难于满足海量数据的场景。

1. 从性能方面来说，由于关系型数据库大多采用 B+ 树类型的索引，在数据量超过阈值的情况下，索引深度的增加也将使得磁盘访问的 IO 次数增加，进而导致查询性能的下降； 同时，高并发访问请求也使得集中式数据库成为系统的最大瓶颈。
2. 从可用性的方面来讲，服务化的无状态性，能够达到较小成本的随意扩容，这必然导致系统的最终压力都落在数据库之上。 而单一的数据节点，或者简单的主从架构，已经越来越难以承担。数据库的可用性，已成为整个系统的关键。
3. 从运维成本方面考虑，当一个数据库实例中的数据达到阈值以上，对于 DBA 的运维压力就会增大。 数据备份和恢复的时间成本都将随着数据量的大小而愈发不可控。一般来讲，单一数据库实例的数据的阈值在 1TB 之内，是比较合理的范围。

数据分片指按照某个维度将存放在单一数据库中的数据分散地存放至多个数据库或表中以达到提升性能瓶颈以及可用性的效果。 数据分片的有效手段是对关系型数据库进行分库和分表。

* 分库和分表均可以有效的避免由数据量超过可承受阈值而产生的查询瓶颈
* 分库还能够用于有效的分散对数据库单点的访问量
* 分表够提供尽量将分布式事务转化为本地事务的可能，一旦涉及到跨库的更新操作，分布式事务往往会使问题变得复杂。

使用多主多从的分片方式，可以有效的避免数据单点，从而提升数据架构的可用性。

什么是分库分表？

用于解决大规模应用中单一数据库容量不足以支持高并发和大数据量的问题。

**垂直分片：**按照业务拆分。比如，电商库 mall\_db，业务拆分后就是 user\_db、order\_db、pay\_db...

**水平分片：**通过字段拆分，比如，用户库 user\_db，分片库后就是 user\_db\_0、user\_db\_1、user\_db\_xx。

什么场景分表？

数据量过大或者数据库表对应的磁盘文件过大。

**Q：多少数据量进行分表？单表 1000w 是否要分表？**

A：假设一个表里 15 个字段，没有特别大的值（不包含 text 或其它超长度的列）数据量超过 5000 万了，依然很丝滑，因为走索引。

真正需要考虑的是：**业务的增长量以及历史数量**。

**Q：物理文件过大，会有什么问题？**

A：会影响公司对数据库表的一个**备份**。数据库表文件过大，也间接证明表数据过大，**增加或删除**字段导致锁表的时间过长。

什么场景分库？

当数据库的连接不够客户端使用时，可以考虑分库或读写分离。如果说当数据库的 QPS 越来越高以及数据量越来越大的时候，就需要考虑分库分表。

**Q：为什么说连接不够用？**

A：假设 MySQL Server 能支持 4000 个数据库连接。我们有 10 个服务，40 个节点，一个节点呢数据库连接池最多 10 个。这样就把一个 MySQL Server 的连接数压榨干净了。

当 MySQL 连接不够用时，可能会报错 "Too many connections" 或者类似的错误。这是因为 MySQL 服务器同时可以处理的连接数量是有限制的，当连接数达到这个限制时，服务器就会拒绝新的连接请求，并返回这个错误消息。

什么场景分库分表？

* **高并发写入场景：**当应用面临高并发的写入请求时，单一数据库可能无法满足写入压力，此时可以将数据按照一定规则拆分到多个数据库中，每个数据库处理部分数据的写入请求，从而提高写入性能。
* **数据量巨大场景：**随着数据量的不断增加，单一数据库的存储和查询性能可能逐渐下降。此时，可以将数据按照一定的规则拆分到多个表中，每个表存储部分数据，从而分散数据的存储压力，提高查询性能。

什么是 ShardingSphere

分布式的数据库生态系统

可以将任意数据库转换为分布式数据库

通过数据分片、弹性伸缩、加密等能力对原有数据库进行增强

设计哲学为 Database Plus，关注如何充分合理地利用数据库的计算和存储能力，而并非实现一个全新的数据库。关注数据库之间的协作多于数据库自身。

ShardingSphere 有两个产品在企业中广泛使用，分别是 ShardingSphere-JDBC 以及 ShardingSphere-Proxy。

ShardingSphere-JDBC

ShardingSphere-JDBC 定位为轻量级 Java 框架，在 Java 的 JDBC 层提供的额外服务。 它使用客户端直连数据库，以 Jar 包形式提供服务，无需额外部署和依赖，可理解为增强版的 JDBC 驱动，完全兼容 JDBC 和各种 ORM 框架。

* 适用于任何基于 JDBC 的 ORM 框架，如：JPA, Hibernate, Mybatis, Spring JDBC Template 或直接使用 JDBC；
* 支持任何第三方的数据库连接池，如：DBCP, C3P0, BoneCP, HikariCP 等；
* 支持任意实现 JDBC 规范的数据库，目前支持 MySQL，PostgreSQL，Oracle，SQLServer 以及任何可使用 JDBC 访问的数据库。

ShardingSphere-Proxy

ShardingSphere-Proxy 定位为透明化的数据库代理端，通过实现数据库二进制协议，对异构语言提供支持。 目前提供 MySQL 和 PostgreSQL 协议，透明化数据库操作，对 DBA 更加友好。

* 向应用程序完全透明，可直接当做 MySQL/PostgreSQL 使用；
* 兼容 MariaDB 等基于 MySQL 协议的数据库，以及 openGauss 等基于 PostgreSQL 协议的数据库；
* 适用于任何兼容 MySQL/PostgreSQL 协议的的客户端，如：MySQL Command Client, MySQL Workbench, Navicat 等。

混合部署

* ShardingSphere-JDBC 采用无中心化架构，与应用程序共享资源，适用于 Java 开发的高性能的轻量级 OLTP 应用；
* ShardingSphere-Proxy 提供静态入口以及异构语言的支持，独立于应用程序部署，适用于 OLAP 应用以及对分片数据库进行管理和运维的场景。

通过混合使用 ShardingSphere-JDBC 和 ShardingSphere-Proxy，并采用同一注册中心统一配置分片策略，能够灵活的搭建适用于各种场景的应用系统，使得架构师更加自由地调整适合于当前业务的最佳系统架构。

**OLTP（Online Transaction Processing，联机事务处理）**

OLTP 就像一个超市收银系统：

* 每次你结账时，系统会记录商品信息、价格、库存等。
* 每个操作（如购买、退货）就是一次“事务”，需要快速完成。
* 这些事务可能很多，但每次处理的数据量较小。

**OLTP 应用的特点**

1. **高并发**：能够同时处理大量用户请求，例如多个用户同时进行购物或查询。
2. **实时性强**：数据处理需要迅速完成，保证用户操作的即时反馈。
3. **数据量小但频繁**：每次事务处理的数据量较小（如单笔订单），但请求频繁。
4. **事务性**：强调 **ACID（原子性、一致性、隔离性、持久性）**，确保数据的正确性和可靠性。

产品功能

| **特性** | 定义 |
| --- | --- |
| **数据分片** | 数据分片，是应对海量数据存储与计算的有效手段。ShardingSphere 基于底层数据库提供分布式数据库解决方案，可以水平扩展计算和存储。 |
| **读写分离** | 读写分离，是应对高压力业务访问的手段。基于对 SQL 语义理解及对底层数据库拓扑感知能力，ShardingSphere 提供灵活的读写流量拆分和读流量负载均衡。 |
| **数据加密** | 数据加密，是保证数据安全的基本手段。ShardingSphere 提供完整、透明、安全、低成本的数据加密解决方案。 |
| **影子库** | 在全链路压测场景下，ShardingSphere 支持不同工作负载下的数据隔离，避免测试数据污染生产环境。 |

设计哲学

设计哲学为 Database Plus，关注如何充分合理地利用数据库的计算和存储能力，而并非实现一个全新的数据库。关注数据库之间的协作多于数据库自身。

数据分片核心概念

准备转黑马。

1. 表

1.2 逻辑表

1.3 真实表

1.4 绑定表

1.5 广播表

1.6 单表

2. 数据节点

2.1 均匀分布

2.2 自定义分布

3. 分片

3.1 分片键

3.2 分片算法

3.3 自动化分片算法

3.4 自定义分片算法

3.5 分片策略

3.6 强制分片路由

4. 行表达式

5. 分布式主键

动手实现数据分表

准备转黑马。

1. 准备数据库&表

2. 开发用户持久层代码

3. 引入 ShardingSphere-JDBC

4. 测试分表效果

针对性学习

参考：拿个offer-12306语雀文档/如何选择适合自己的学习方式/面试冲刺/4.针对性学习

* 2025.02.04 手摸手之从注册用户接口开始
  + 这篇相当于一个目录索引，导向了很多分支
* 2025.02.05 手摸手之实现敏感信息加密存储
  + 核心技术文档-如何防止用户敏感数据泄露 shardingsphere文档
* 2025.02.18 手摸手之用户敏感信息展示脱敏
* 2025.02.18 手摸手之实现列车购票流程
  + 手摸手之实现用户购票责任链验证
  + 核心技术文档-从根上理解Redis分布式锁演进架构
* 2025.02.20
  + 手摸手之订单如何分库分表（代码待细看）
  + 手摸手之乘车人本人车票订单查看
  + 订单延时关闭功能技术选型（最后选择RocketMQ实现）
* 2025.02.20 手摸手之实现v2版本列车购票流程（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？

面试实战

参考：【大二进美团二面，疯狂拷打项目，面试官直言12306见多了？如何让面试官眼前一亮】<https://www.bilibili.com/video/BV1Dn4y1X7aD?vd_source=7341e06df79108c355b8df461a20c071>

**背景：**

12306铁路购票系统，在秋招前进行练手，选择开源项目，模仿12306完成了核心的相关功能。

**五大模块：**

网管模块：JWT令牌校验、请求路由转发

用户模块：登陆注册、增删改查

购票模块（最核心）最细节）：查票购票

订单模块、支付模块：购票后订单、取消支付

**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。