美团业务正确性校验平台BCP的设计与实践

美团基础架构部

CODE A BETTER LIFE

一 行 代 码　亿 万 生 活

个人简介

叶明

基础架构部-服务保障组

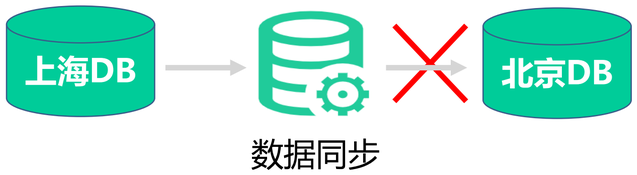
2017 年加入美团，先后参与过Squirrel分布式缓存、DTS数据传输服务、BCP业务正确性校验平台、SWAN分布式事务等中间件建设。目前，主要负责BCP和SWAN相关的研发工作

目录

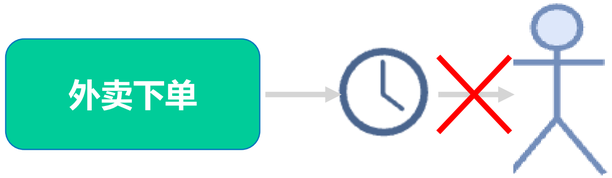
BCP产生背景-数据异常问题

分布式环境下，RPC调用超时、MQ丢消息、存储组件读写失败难以避免，导致服务内部/服务之间出现数据问题

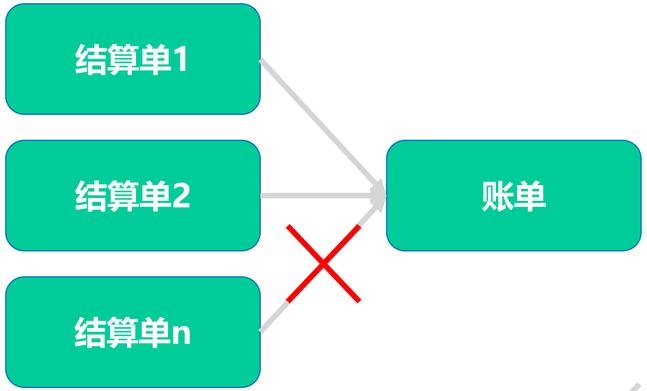
一致性：异地多活场景，同步出现问题导致数据不一致



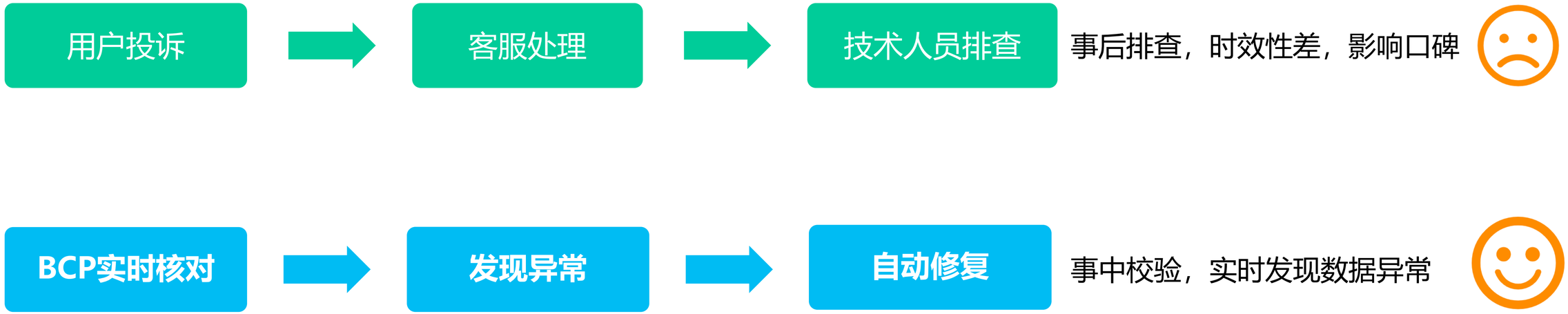
时效性：系统原因用户下单后没有准时收到外卖



正确性：核对账单，少了一笔费用导致对账不平衡



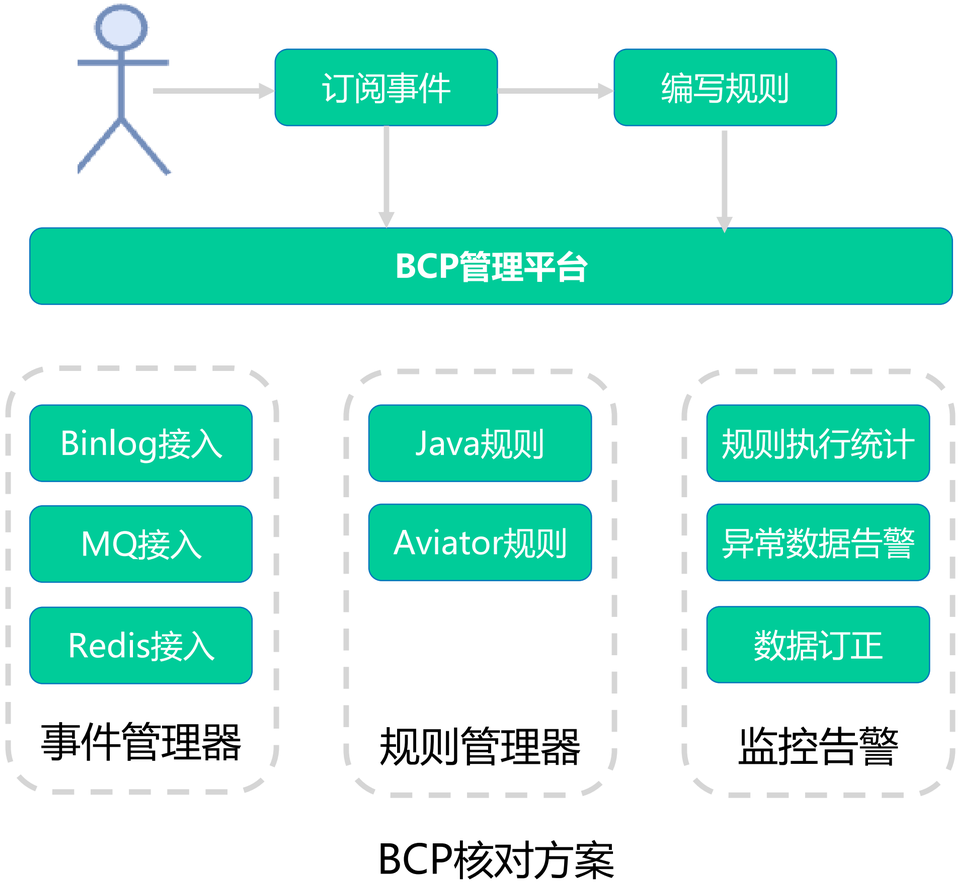
BCP产生背景-数据问题处理流程



BCP是什么

BCP（Business Check Platform）美团业务正确性校验平台

标准化数据源接入，基于事件触发核对规则的执行，实时发现异常数据并及时订正



BCP核对方案

BCP接入规模

3000+

规则个数

200亿+/天

规则核对次数

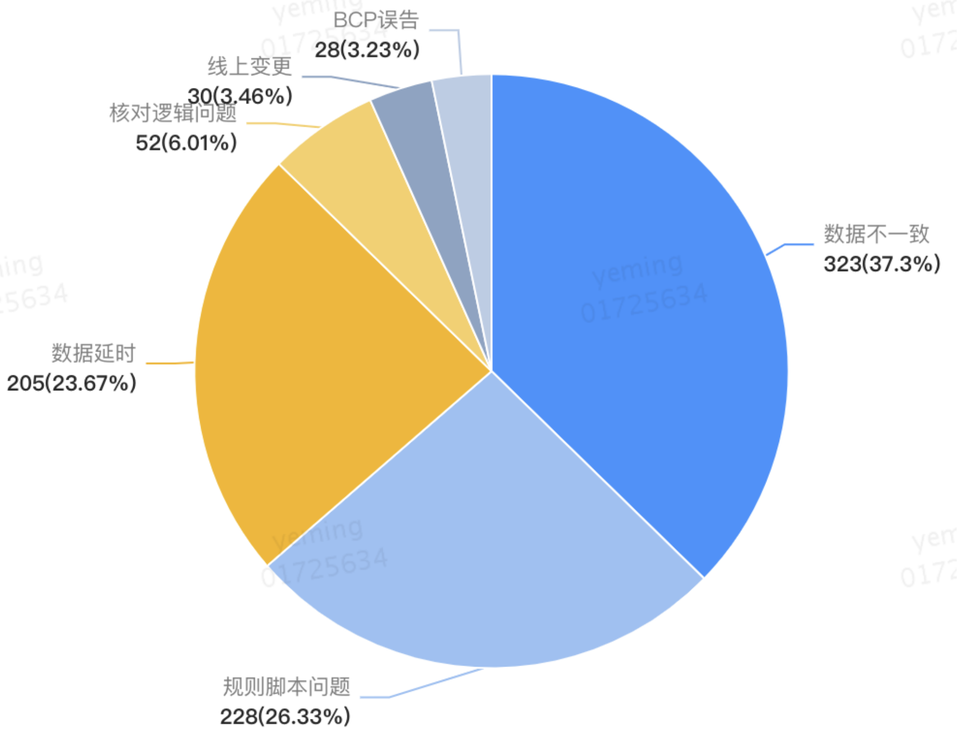
400亿+/天

Binlog订阅量

50亿+/天

MQ订阅量

多次帮助业务发现数据异常



目录

BCP技术设计

整体架构

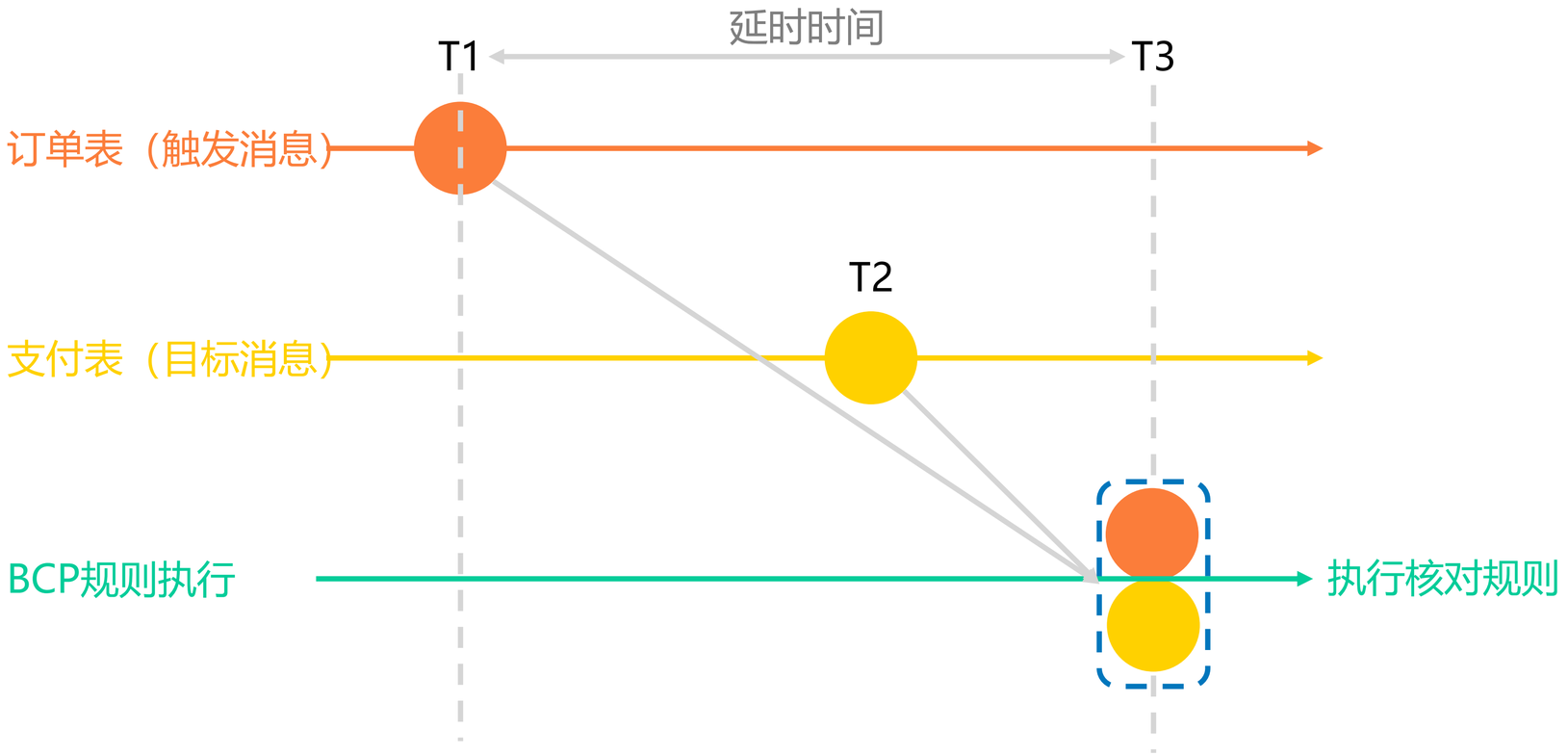
稳定性保障设计

系统处理过程
整体架构

BCP系统处理流程

购买逻辑涉及更新订单和支付单状态，假设我们需要校验用户下单 60s 内是否完成支付且金额是否一致。我会从系统（宏观）和规则（微观）两个层面叙述系统行为。

BCP规则执行流程



检验每一笔交易60s内是否完成支付

① 订单表产生Binlog，订单 ID 为匹配键

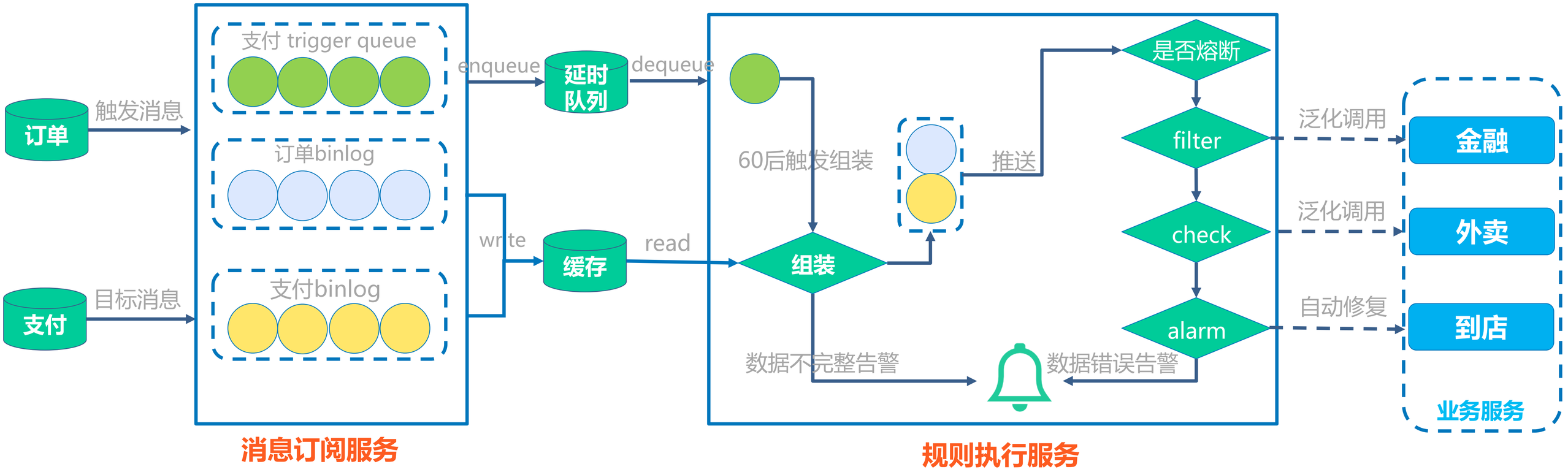
②支付表产生Binlog，订单 ID 为匹配键

③60s之后根据匹配键触发消息组装

order.id == payment.id &&
order.amount == payment.amount

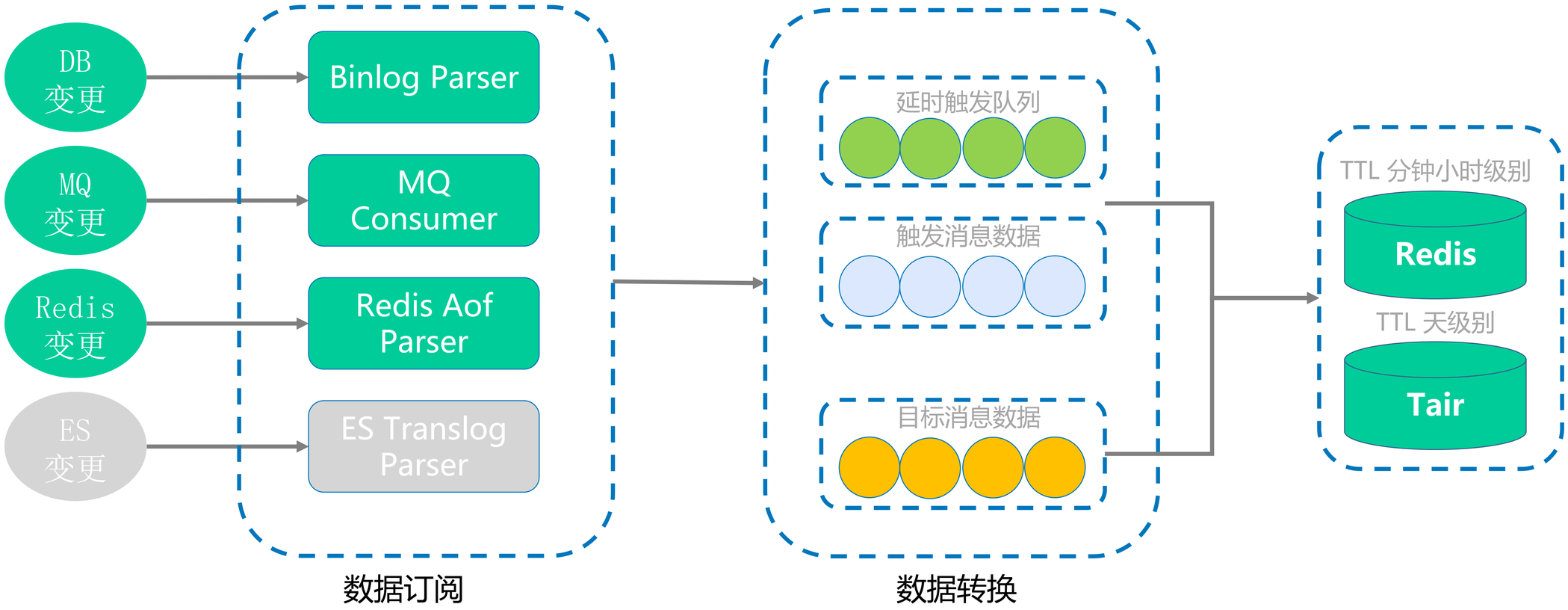
BCP整体架构

BCP核心服务包括**消息订阅服务**和**规则执行**服务，基于**事件驱动**的方式进行实时校验。



BCP整体架构-消息订阅服务

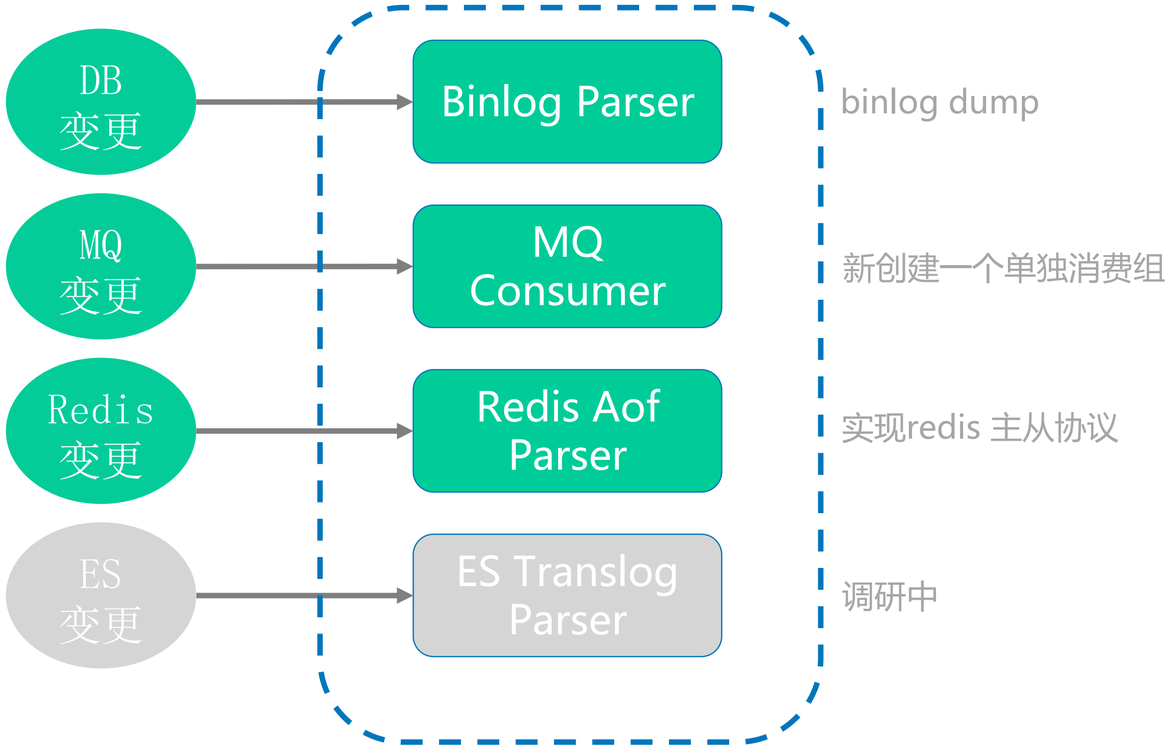
消息订阅服务主要包含：数据订阅、数据转换模块



消息订阅服务-增量订阅组件

标准化的数据订阅CDC组件

对接Binlog、MQ、Redis、ES



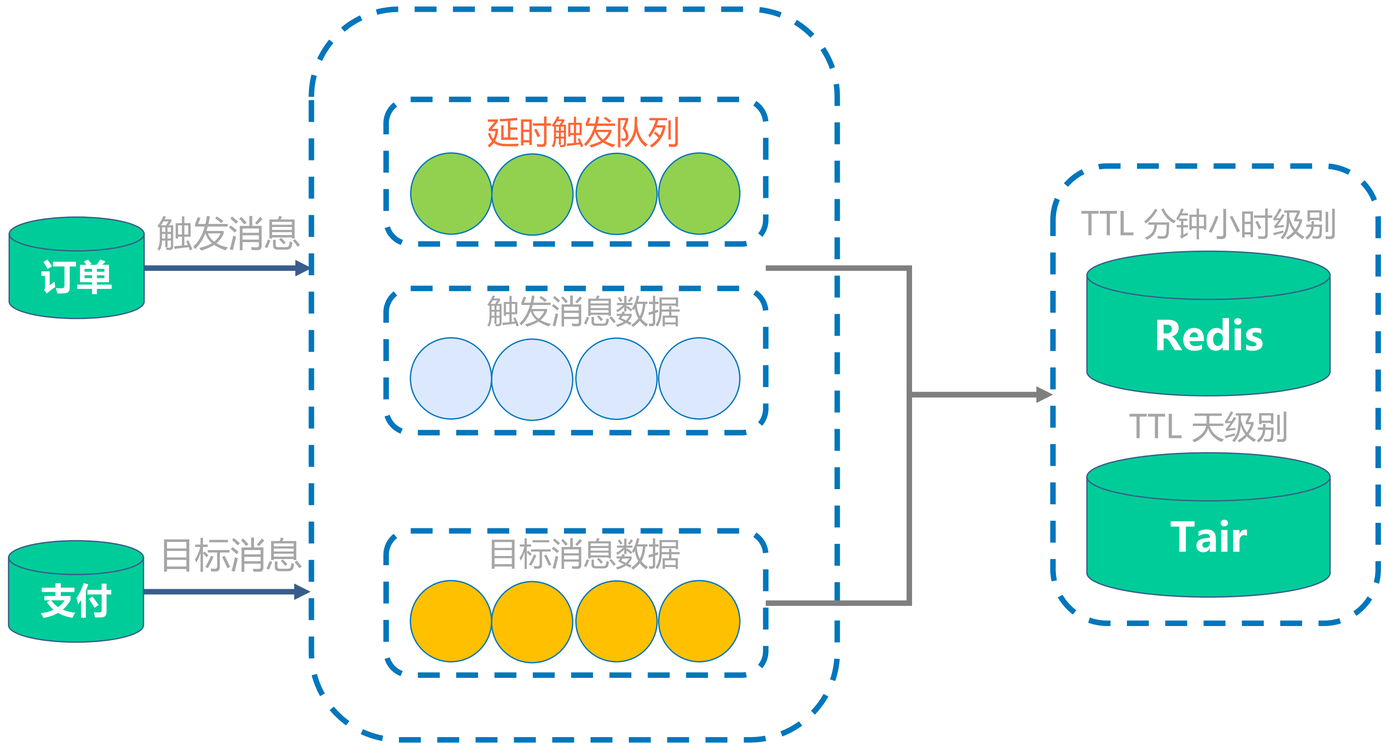
消息订阅服务-数据转换

延时队列：基于触发消息时间戳生成

触发消息：用来驱动规则校验的事件

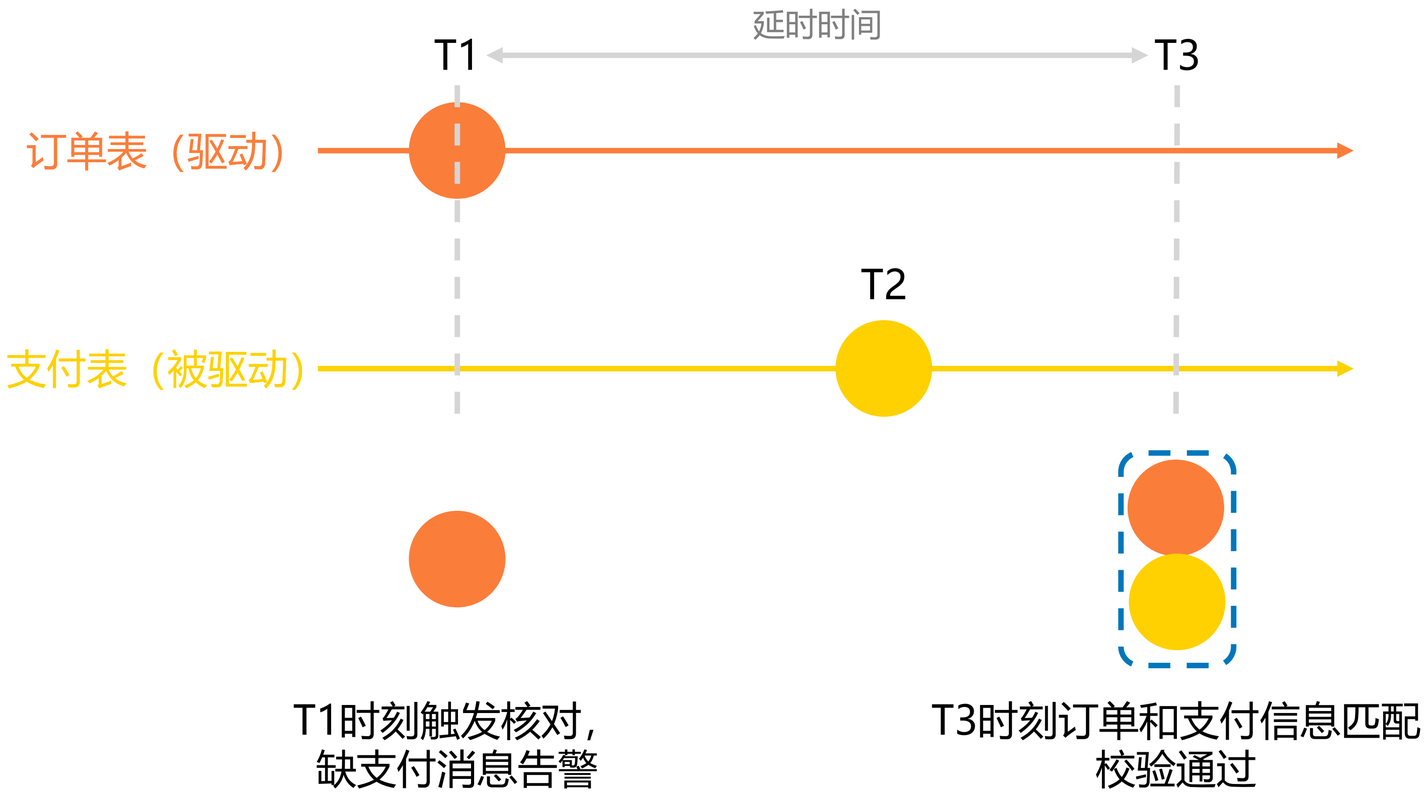
目标消息：被驱动事件

支持双向校验：订单和支付双向触发



分布式延时队列-背景

双流核对的场景，需要设计一个延时队列，将触发消息先投递到队列，一定延时时间以后再执行



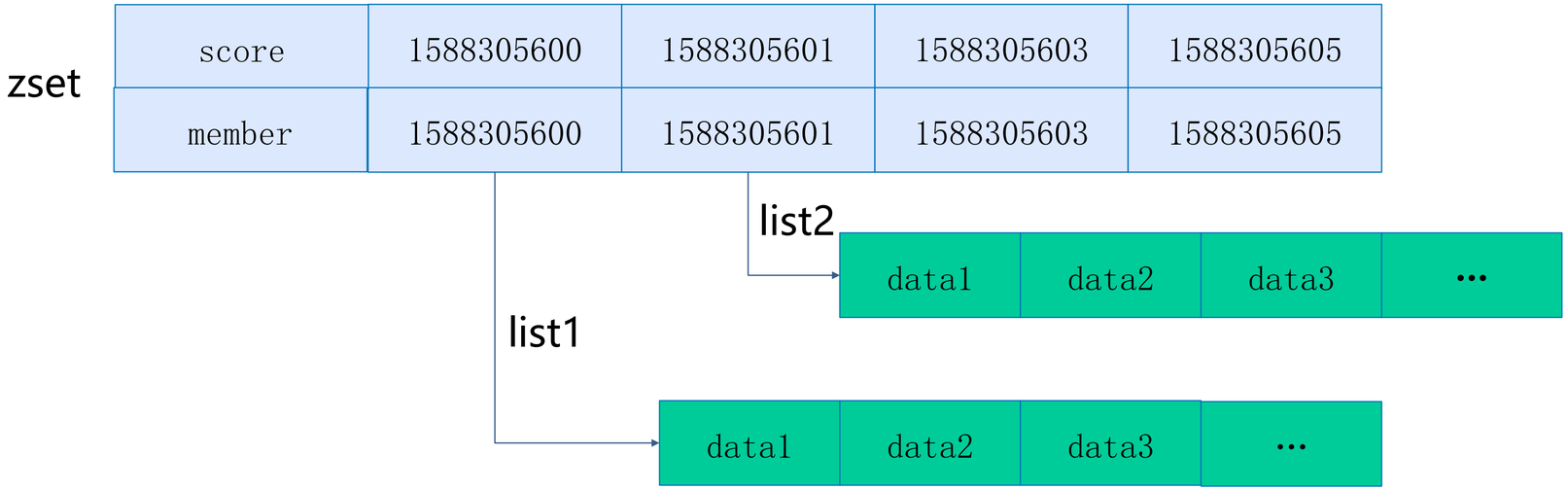
分布式延时队列-整体设计

整体设计思路：基于Redis zset和list实现分布式延时队列

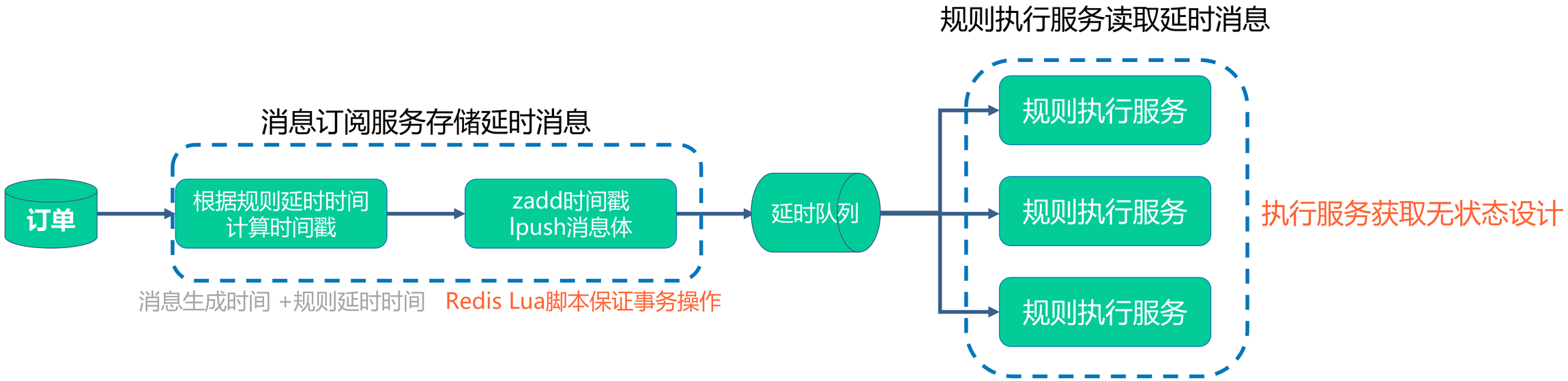
zset：存储时间戳，时间戳有序递增

list：触发消息和目标消息体、规则id、触发消息匹配键值(订单id)、

拆分时间戳和消息体，避免Redis zset大Value问题



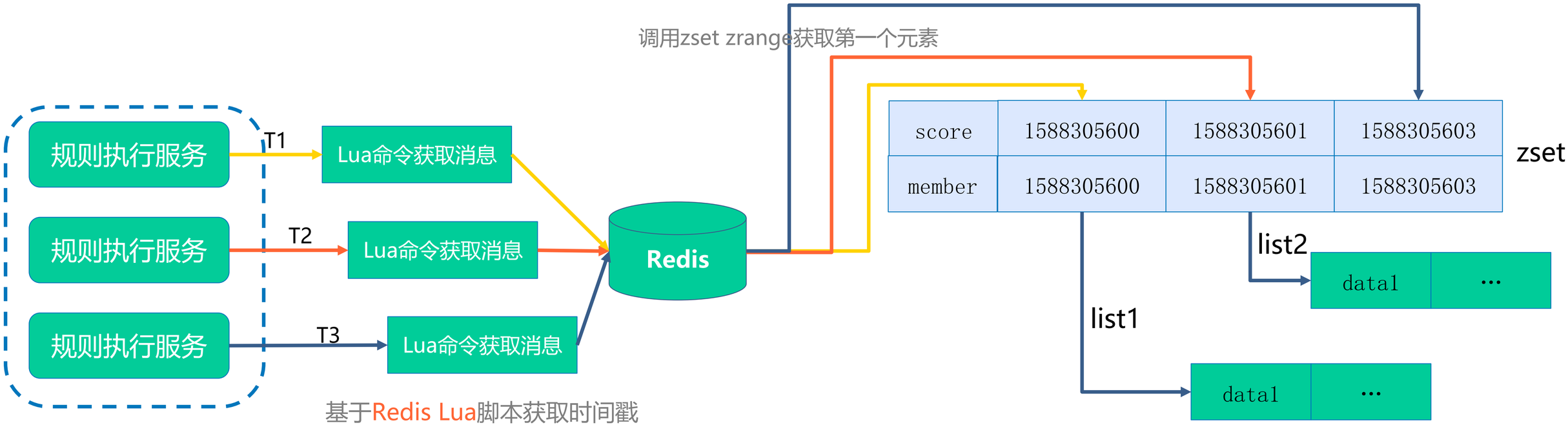
分布式延时队列-整体执行过程



分布式延时队列-读取过程

如何保证执行服务负载均衡和不重复获取时间戳

实现思路：中心时钟设计、Redis Lua脚本、Redis单线程



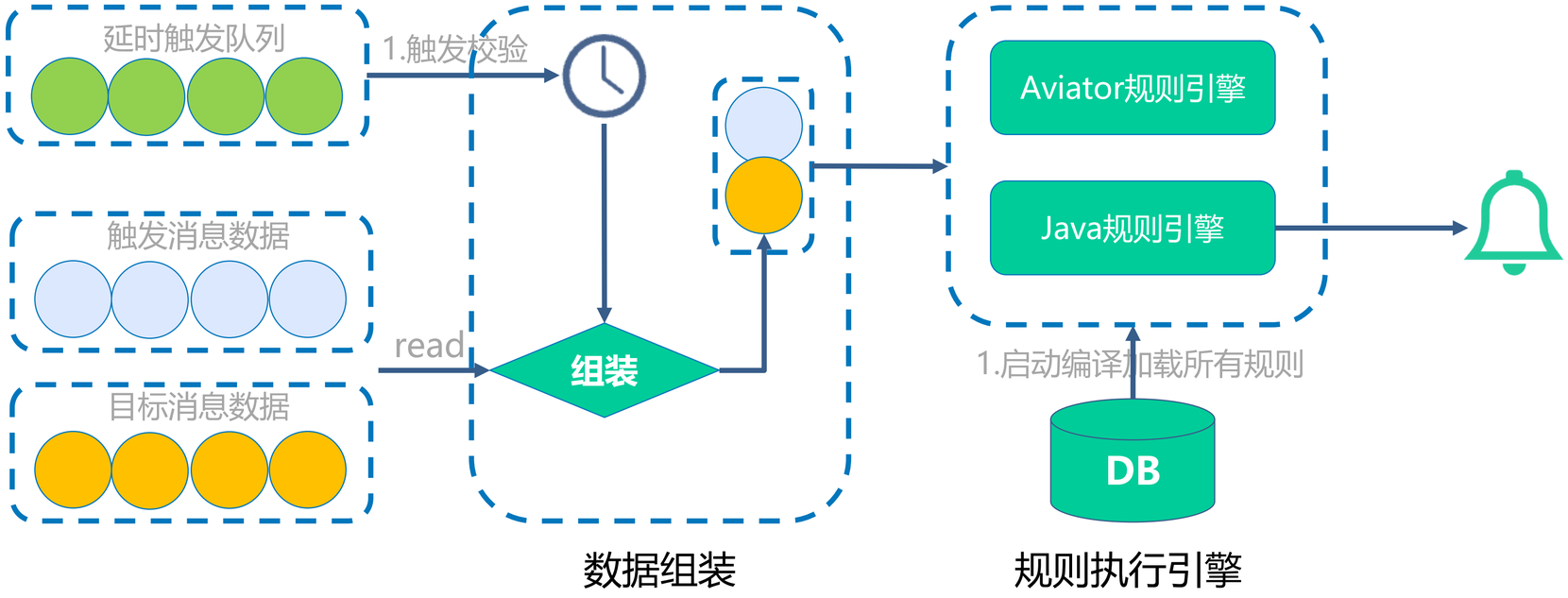
基于Redis Lua脚本获取时间戳

1. 判断zset中首元素是否大于Redis系统时间
2. 根据时间戳获取对应的member
3. 删除zset中首元素

智能调速策略尽最大可能平衡执行机器负载

BCP整体架构-规则执行服务

规则执行服务主要包含2个模块：数据组装、规则执行引擎

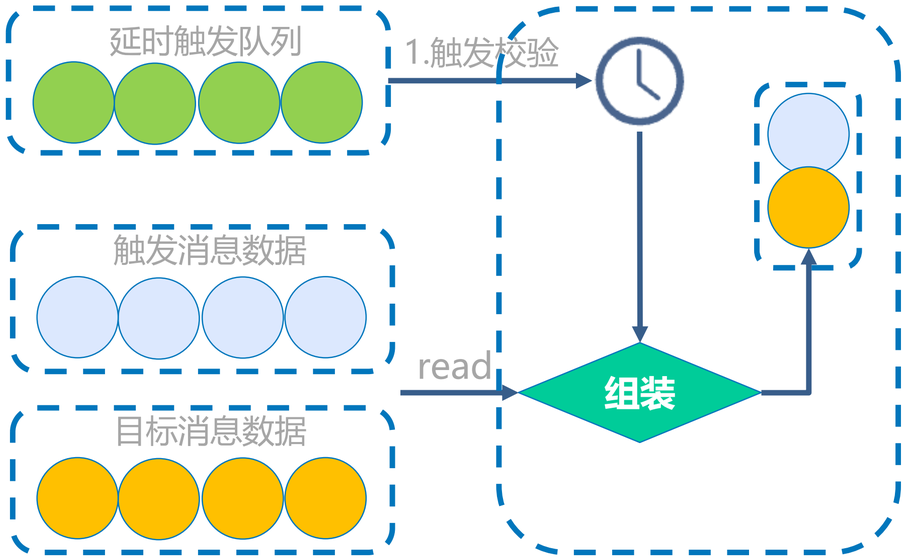


规则执行服务-数据组装

轮询从延时队列中获取待校验消息

根据消息中的匹配键获取触发和目标消息

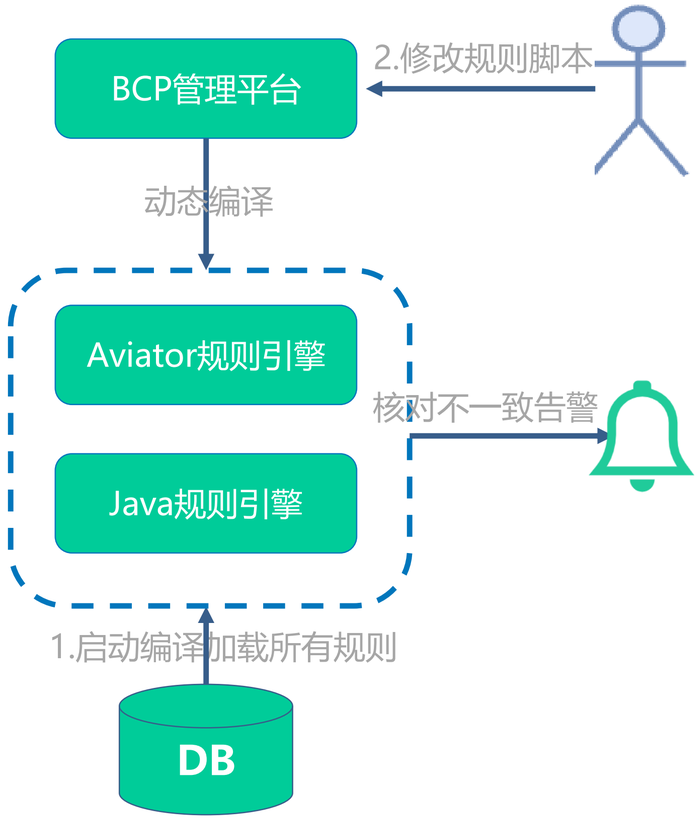
根据消息中的规则id获取规则脚本交由执行引擎执行



规则执行服务-执行引擎

动态编译规则脚本：Aviator、Java

校验不一致数据实时告警



规则执行引擎-Aviator

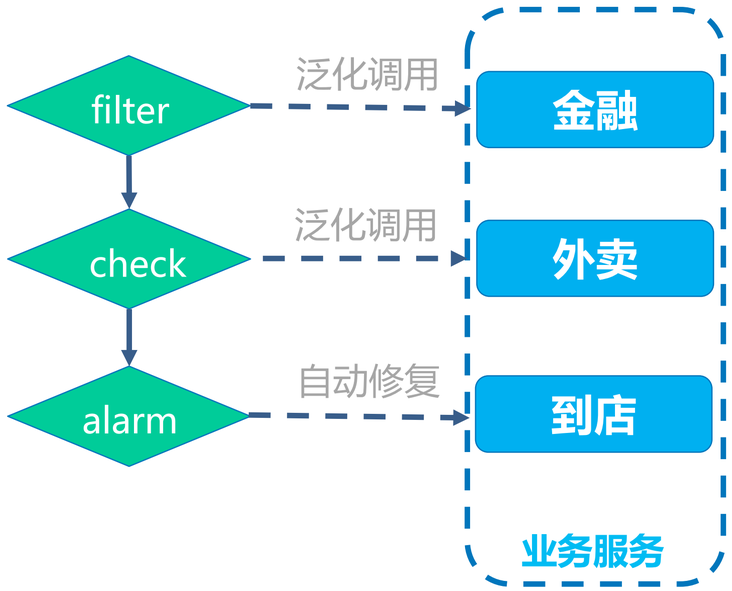
Aviator是一门高性能、轻量级寄宿于 JVM 之上的脚本语言。适用核对场景：

1. 触发消息和目标消息是否同时存在
2. 比较触发消息和目标消息的字段值的关系：是否相等，是否包含某些特定的字符串


```
triggerMsg.id.newValue == targetMsg.id.newValue &&
triggerMsg.column1.newValue == targetMsg.column2.newValue
```

规则执行引擎-Java

适用于复杂定制化的核对逻辑

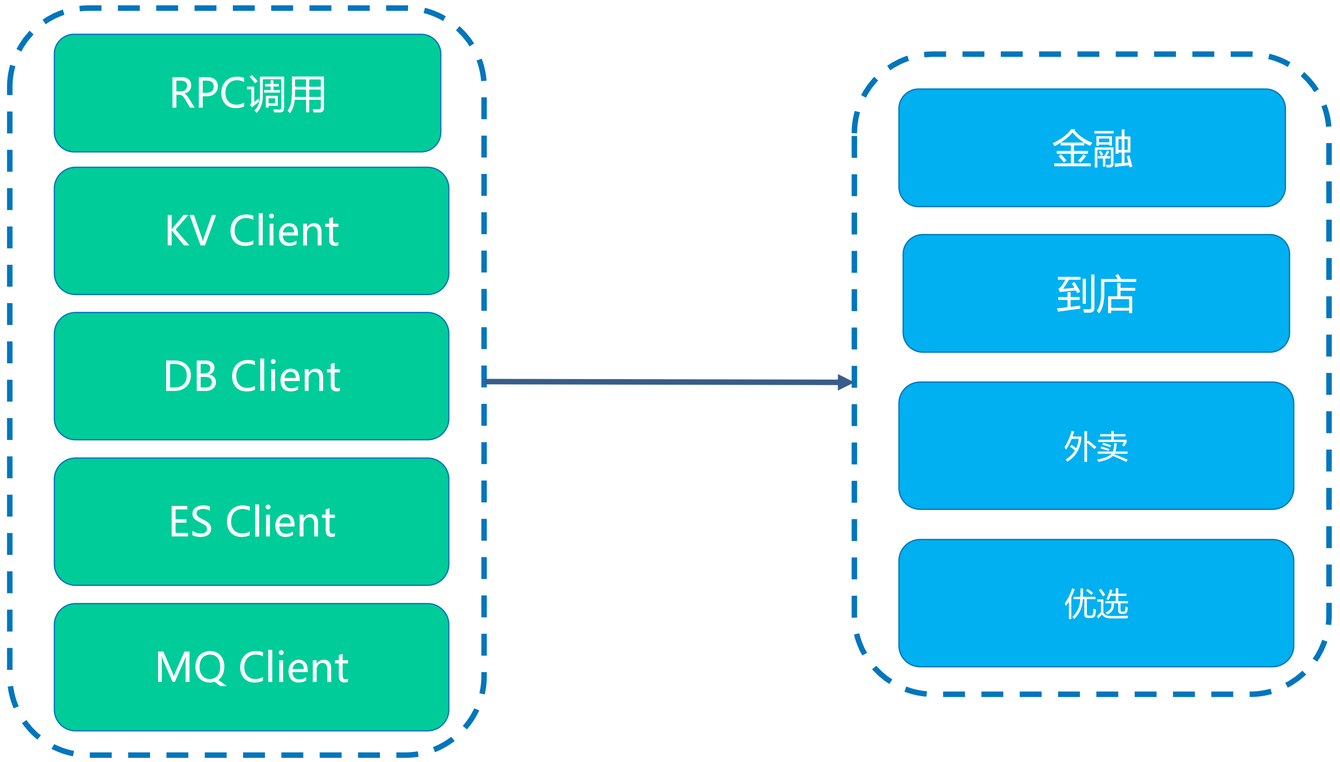



```java
public  class DefaultRuleRunner implements RuleRunner<RawData> {

    /**
     * 过滤规则
     * @param triggerData 触发数据：触发数据源的数据
     * @param targetData 目标数据：根据触发数据源的设置的key对应的其他源数据源的数据；单流校验为空，多流校验则为所有数据源数据除去触发数据
     * @return false:该消息不用校验，true:该消息需要校验，后续继续调用check，alarm方法
     * 注意：代码内部不要修改RawData及其内部引用所指向的数据，如需要修改，请自行copy RawData对象修改
     */
    @Override
    public boolean filter(RawData triggerData, RawData... targetData) throws Exception {
        //默认所有消息都校验
        return true;
    }

    /**
     * 检测规则
     * @param triggerData 触发数据：触发数据源的数据，即填写的消息1，但如果是双向校验，则两个流都会触发校验，因此triggerData可能是其中任一
     * @param targetData 目标数据：根据触发数据源的设置的key对应的其他源数据源的数据；单流校验为空，多流校验则为所有数据源数据除去触发数据
     *                   单流时targetData为length为0的数组；多流时如果匹配阶段没有匹配上，targetData为length为0的数组
     * @return 为null则不进行告警，非null则调用alarm，并把该方法的返回结果作为参数checkResult传入alarm方法
     * 注意：代码内部不要修改RawData及其内部引用所指向的数据，如需要修改，请自行copy RawData对象修改
     */
    @Override
    public String check(RawData triggerData, RawData... targetData) throws Exception {
        //默认都校验通过
        return null;
    }

    /** 默认情况下alarm方法会自动发送大象告警(不要覆盖该方法)，如需通过mafka或者泛化调用等方式请复写alarm方法
     * @param checkResult 是check方法的返回结果，传入alarm方便进行告警
     * @param triggerData 触发数据：触发数据源的数据
     * @param targetData 目标数据：根据触发数据源的设置的key对应的其他源数据源的数据；单流校验为空，多流校验则为所有数据源数据除去触发数据
     * 注意：代码内部不要修改RawData及其内部引用所指向的数据，如需要修改，请自行copy RawData对象修改
     */
    @Override
    public void alarm(String checkResult,RawData triggerData, RawData... targetData) throws Exception {
        //通过大象公众号发送告警消息，收件人为告警配置中的告警接收人。
        DXUtil.sendAlarm(checkResult);
    }
}
```

Java规则执行引擎-常用中间件sdk

触发和目标消息本身不足以完成校验，某些核对规则需要RPC调用或者访问第三方存储，核对结果希望通过MQ通知下游

缓存常用组件实例，提供熔断、限流等功能保护第三方服务或者/存储



BCP技术设计

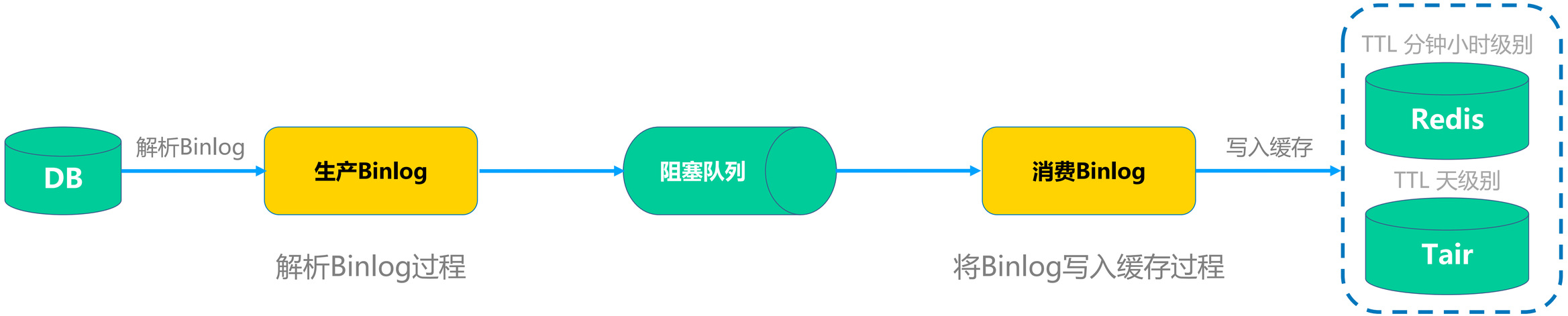
整体架构

稳定性保障设计

执行服务无状态、水平扩容
订阅服务有状态、高性能、高可用、负载均衡设计

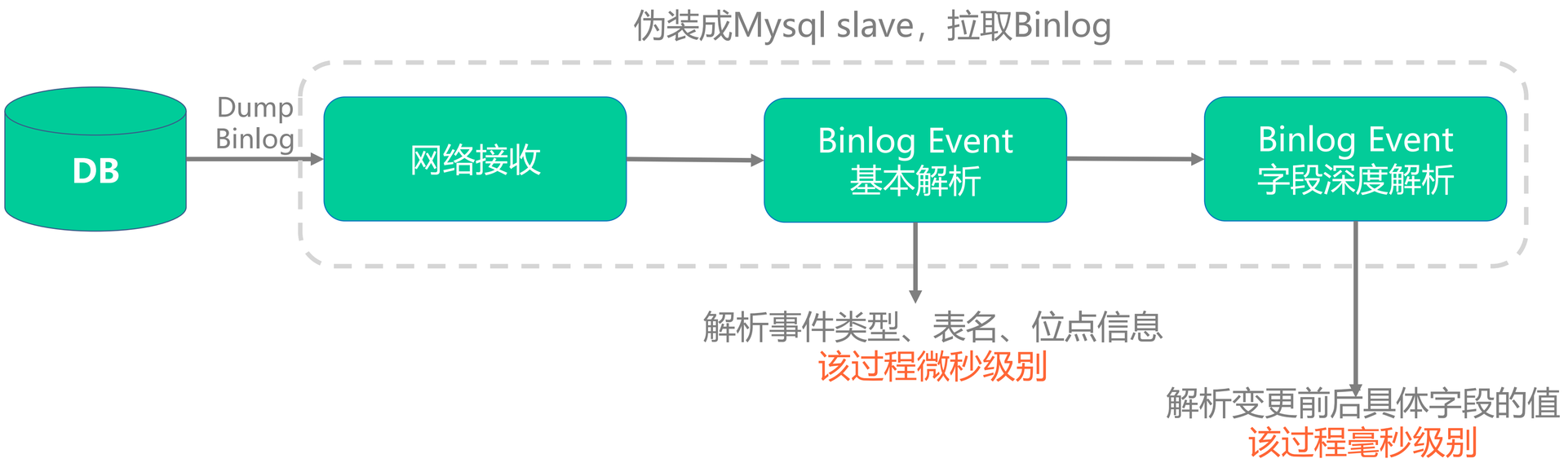
服务高性能设计

生产消费者模型，生产Binlog或者消费Binlog过慢都会导致消息出现延时，一旦超过规则延时时间就会导致规则误告。



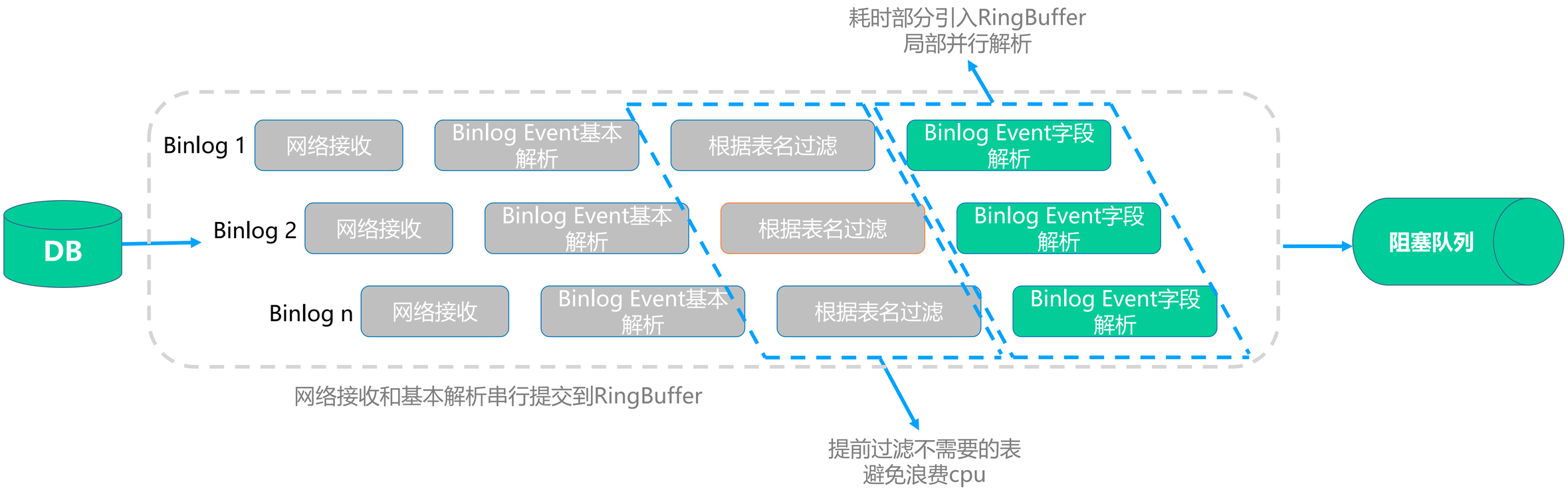
生产Binlog-串行Dump性能瓶颈

常规的Binlog Dump过程采用单线程拉取解析Binlog，吞吐量低



生产Binlog-局部并行化

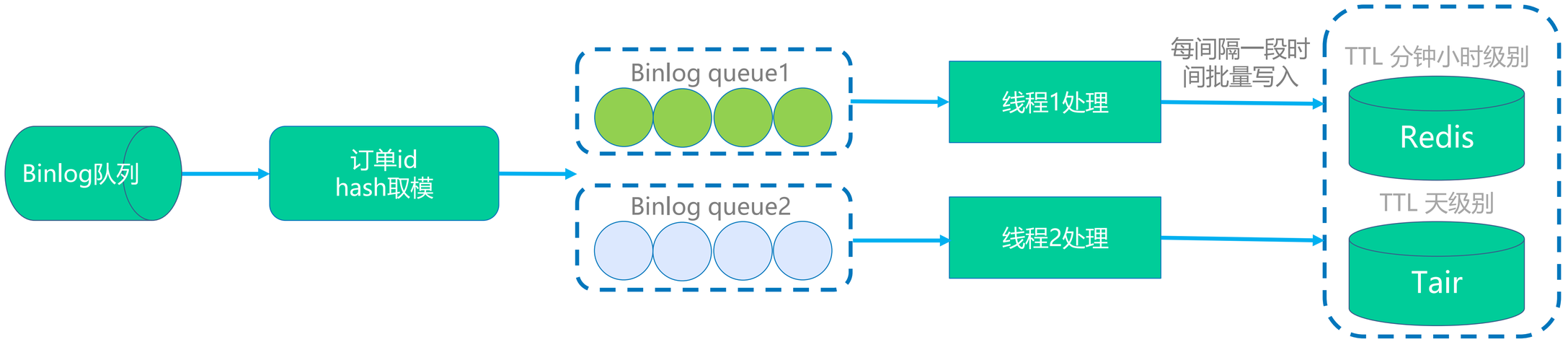
单线程拉取解析Binlog，Event字段解析慢导致瓶颈? → Event深度解析并行化处理 → 如何保证最终有序性?



服务高性能设计-消费Binlog

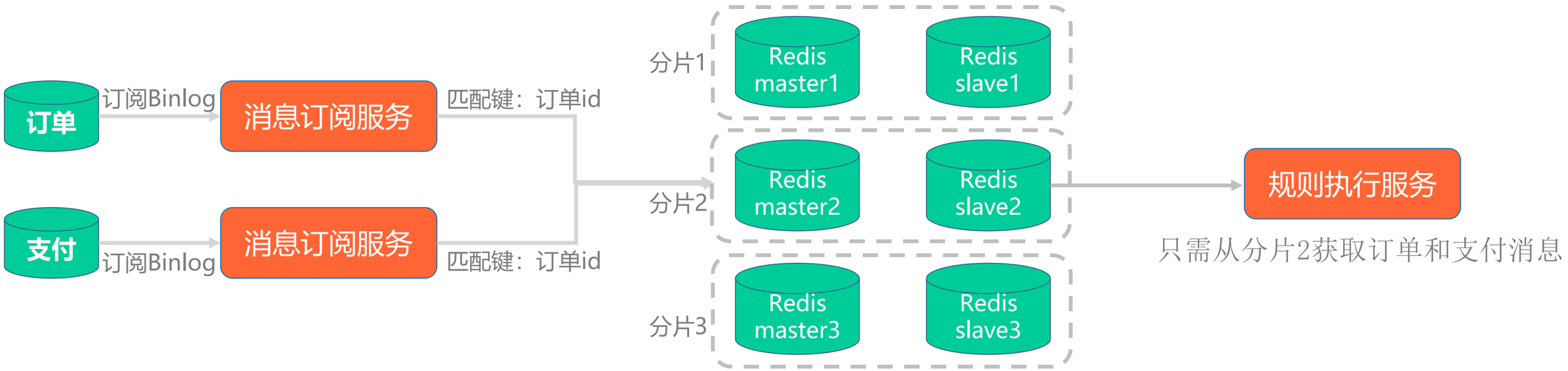
消费Binlog过程主要是从阻塞队列中获取将解析好的Binlog及时写入Redis缓存

优化点：基于业务主键hash多线程处理、非阻塞方式批量提交



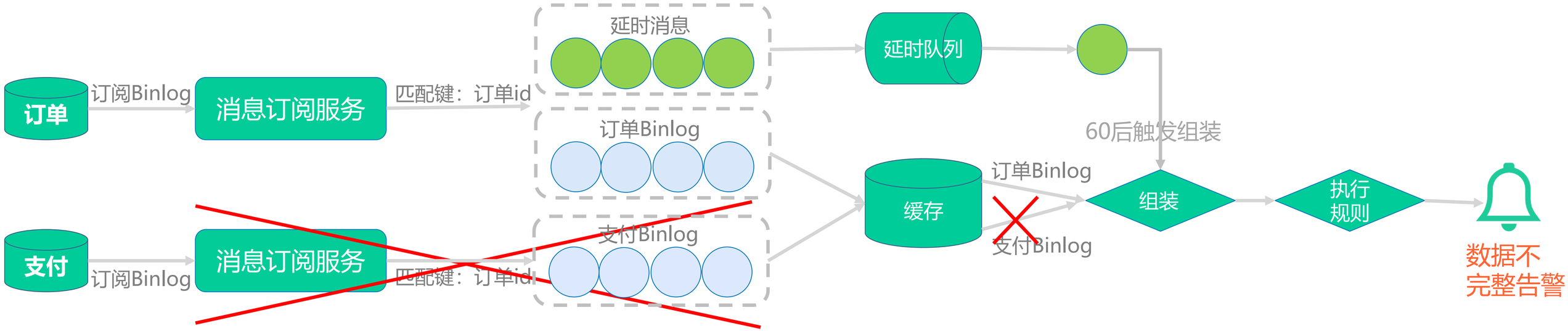
服务高性能设计-减少网络IO次数

基于**Redis HashTag（类似Kafka Partition）** Redis key： {订单id}+key_suffix，Redis保证相同id的订单写入同一Redis节点



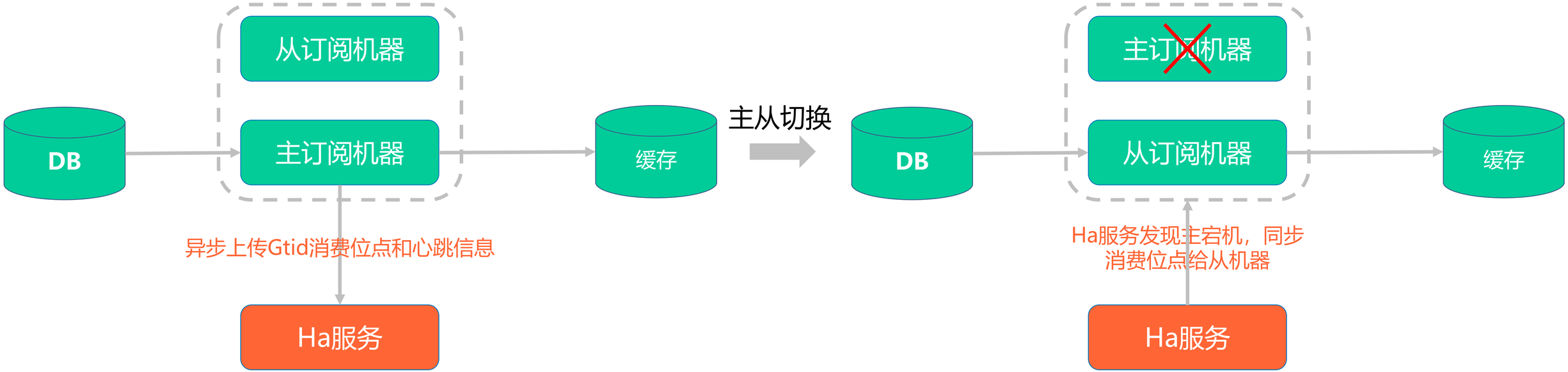
服务高可用设计-订阅服务宕机问题

订阅支付库的订阅机器宕机，无法拉取Binlog写入缓存，造成组装模块组装失败，最终由于缺失支付Binlog导致告警



消息订阅服务高可用设计-宕机切换

高可用设计思路：主从架构的方案保证消息处理的 不重、不漏、低延时



主定时上报消费的位点信息，切换时候从拉取主消费的位点信息开始订阅，保证不漏

主从之前基于zk选主，同时订阅Binlog基于Gtid位点订阅，保证切换低延时

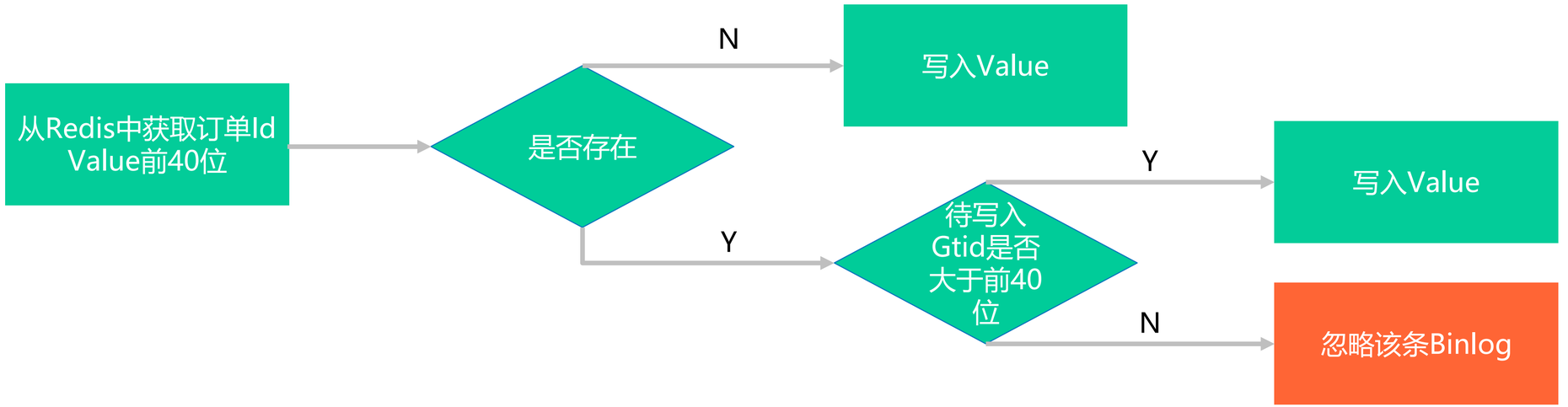
跟MQ类似，发生位点信息交互时很难保证主宕机前把所有位点上报完成，因此从消费时会重复消费一部分Binlog，如何保证不重?

消息订阅服务消费过程幂等设计

基于Redis Lua脚本实现CAS操作，保证永远只会写入Gtid较大的Binlog：

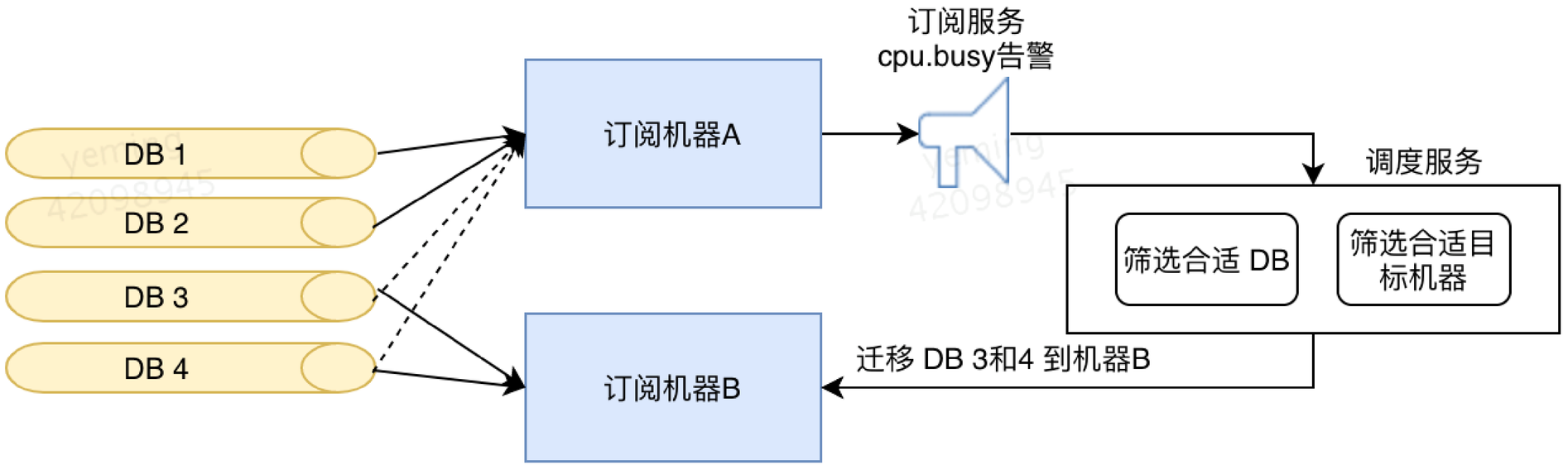
Redis Key：订单Id

Redis Value：40位Gtid+toByte(Binlog本身)



消息订阅服务负载动态均衡设计

如何解决DB流量突增导致的机器负载飙高，避免单机性能瓶颈问题



目录

CONTENTS
目　　录

最佳实践-数据一致性校验

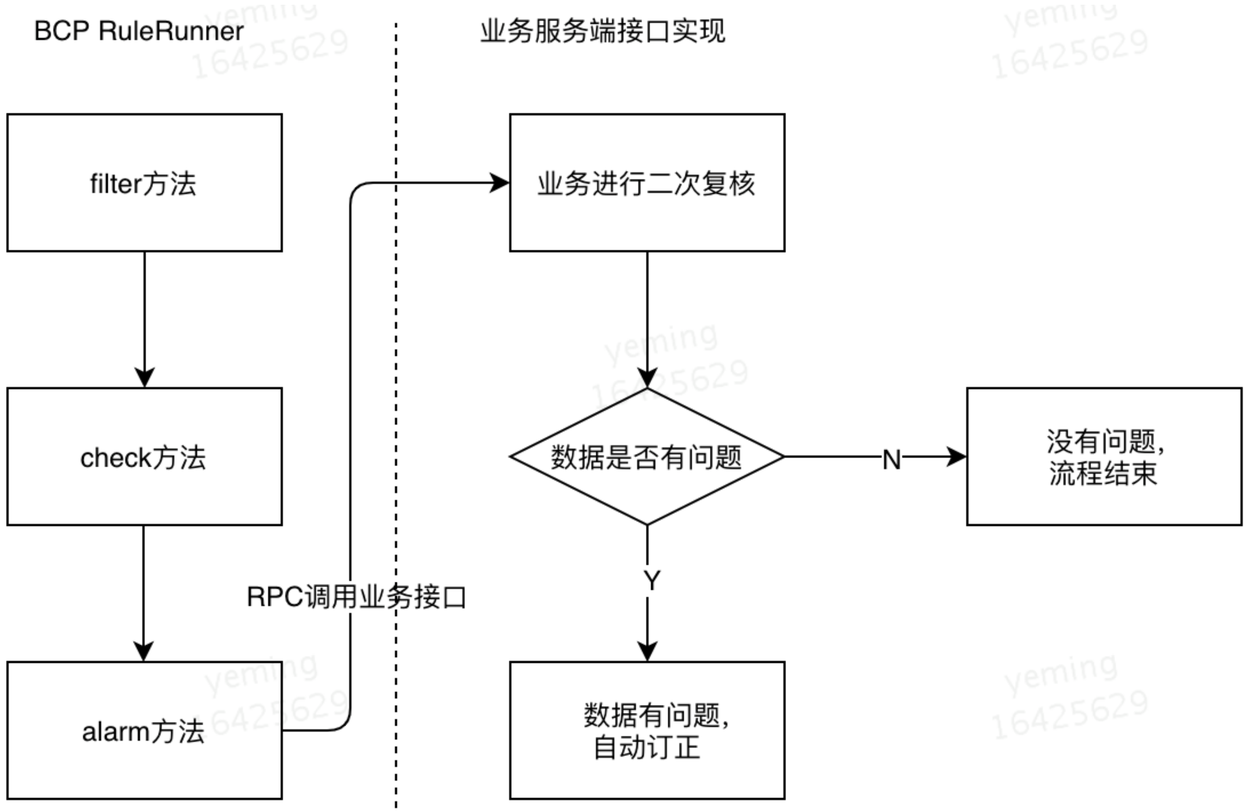
触发消息和目标消息订阅不同的表，使用Aviator表达式判断字段值是否一致

最佳实践-数据时效性校验

触发和目标消息订阅相同的表，校验zcm_acquirer_register表的任务任务状态500s后是否能从1变成7或者10

最佳实践-数据自动订正

覆写BCP的alarm方法，收到告警后调用业务提供数据二次复核接口，确认数据有误进行自动订正



目录

未来规划

新的CDC数据源接入

ES增量变更接入
Blade(Tidb)增量变更接入

离线核对支持

支持文件核对
T+D，T+H核对

基础架构 - Java技术专家/资深工程师

岗位职责

1. 负责美团分布式配置系统、稳定性保障组件、混沌工程、应用容器、业务正确性校验、分布式事务等中间件产品设计与研发工作，不断提升服务稳定性和完善系统功能，满足业务多样化的应用场景。
2. 参与混沌工程领域和稳定性保障能力前沿技术的调研选型，并在项目中落地与推广。
3. 负责设计开发高效的自动化运维平台，提升运维效率；应对突发场景，能够快速发现问题、定位问题和解决问题。

招聘：基础架构 - Java技术专家/资深工程师
邮箱：yeming@meituan.com

更多技术干货
欢迎关注“美团技术团队”

美团 美团

CODE A BETTER LIFE
一行代码 亿万生活

Q&A