Day02

奈学教育

01 Nacos服务注册与发现

02 Sentinel服务治理和流量控制应对流量洪峰

03 RocketMQ 实现瞬时海量订单数据的平滑处理

04 分布式事务解决方案

01.Nacos注册中心深入分析

01.Nacos注册中心深入分析

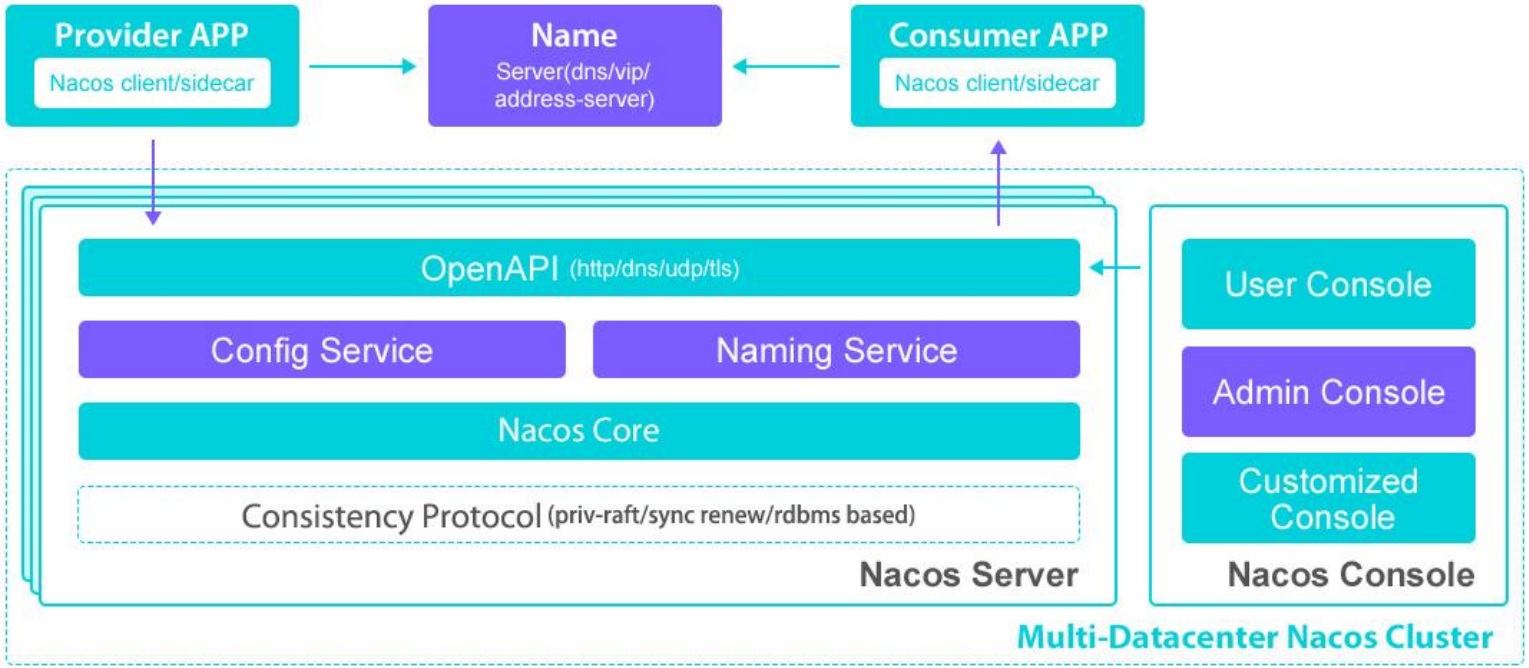
奈学教育

Nacos

Dubbo 生态系统中的注册中心实现

Nacos注册中心功能分析

- 服务注册与健康检查
- 数据模型
- 数据一致性保障



01.Nacos注册中心深入分析

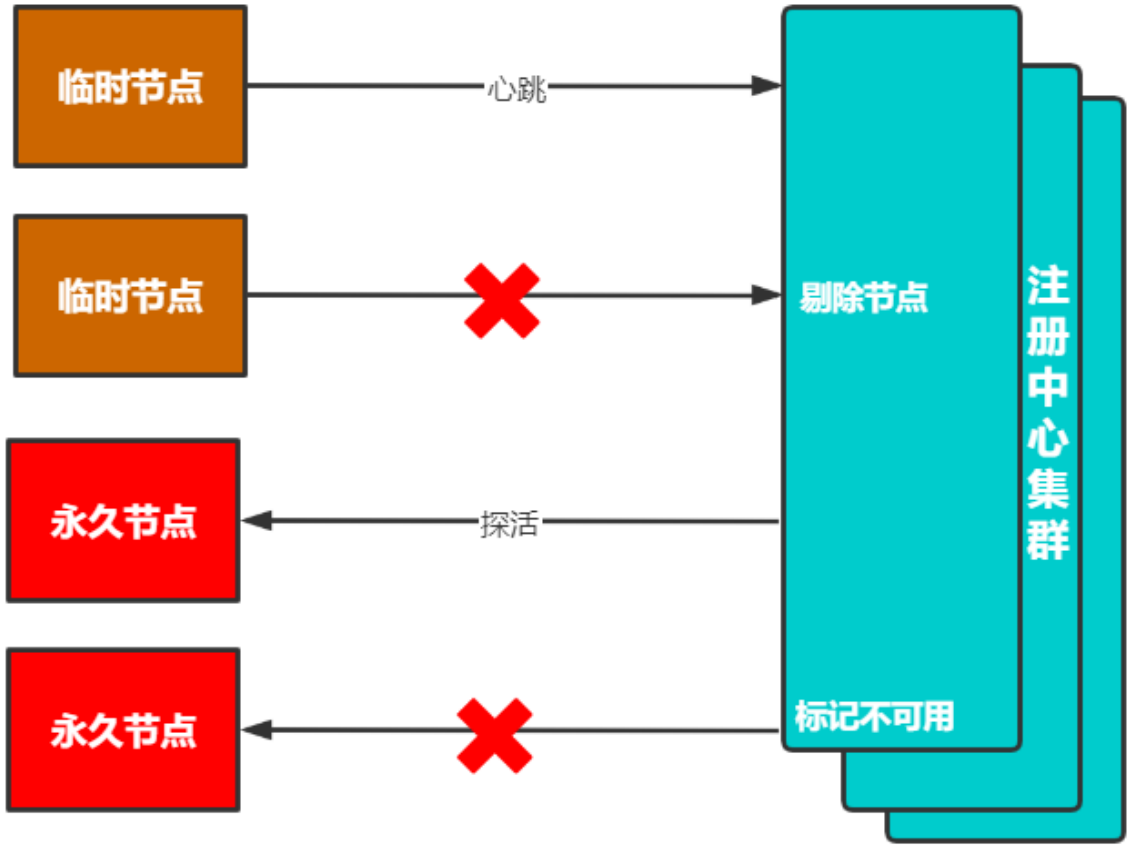
奈学教育

Nacos

Dubbo 生态系统中的注册中心实现

Nacos注册中心功能分析

- 健康检查
  - 临时节点：心跳注册
  - 持久化节点：tcp/http探测
- 数据模型
- 数据一致性保障



临时实例使用心跳上报方式维持活性
5秒上报
15秒标记不健康
30秒剔除

01.Nacos注册中心深入分析

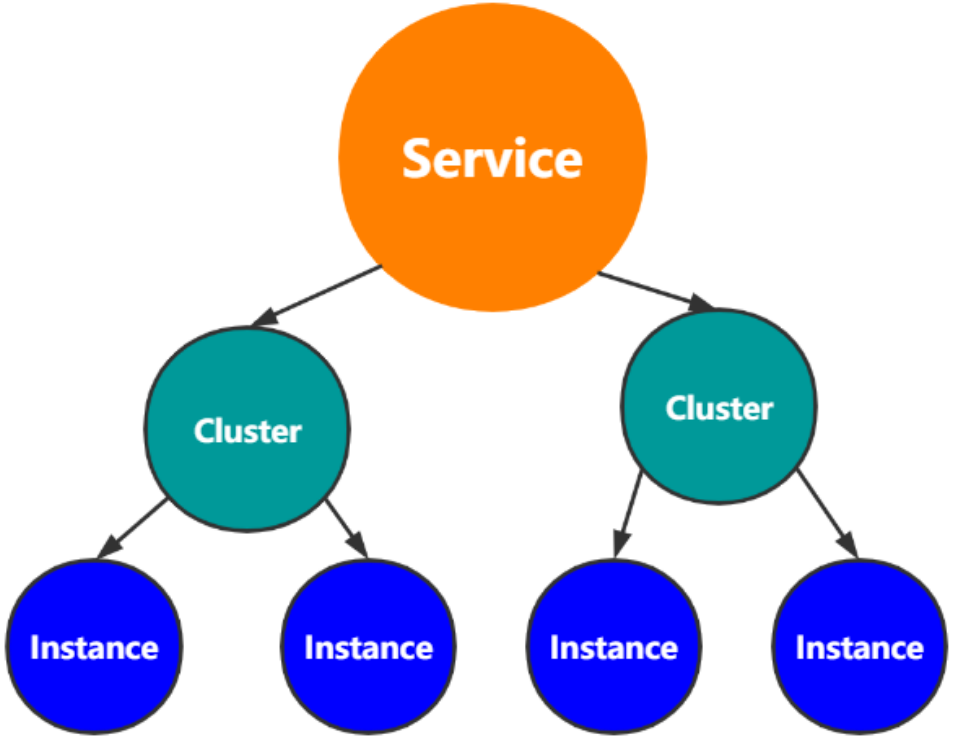
奈学教育

Nacos

Dubbo 生态系统中的注册中心实现

Nacos功能分析

- 服务注册与健康检查
- **数据模型**
  - **数据存储**
  - 数据隔离
- 数据一致性保障



IP地址、端口、健康状态、TTL、权重

01.Nacos注册中心深入分析

NX 奈学教育

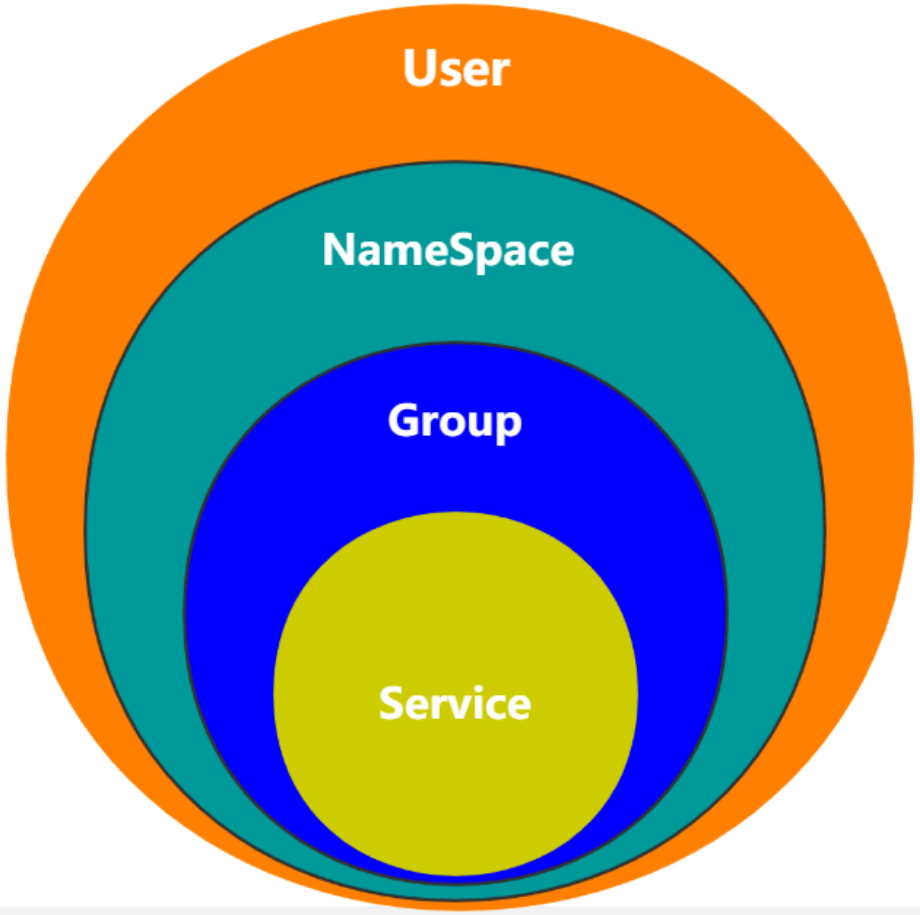
Nacos

Dubbo 生态系统中的注册中心实现

Nacos功能分析

- 服务注册与健康检查
- **数据模型**
  - 数据存储
  - **数据隔离**
- 数据一致性保障

四层数据隔离
账号
命名空间指定集群
分组及服务名标识服务



01.Nacos注册中心深入分析

奈学教育

Nacos

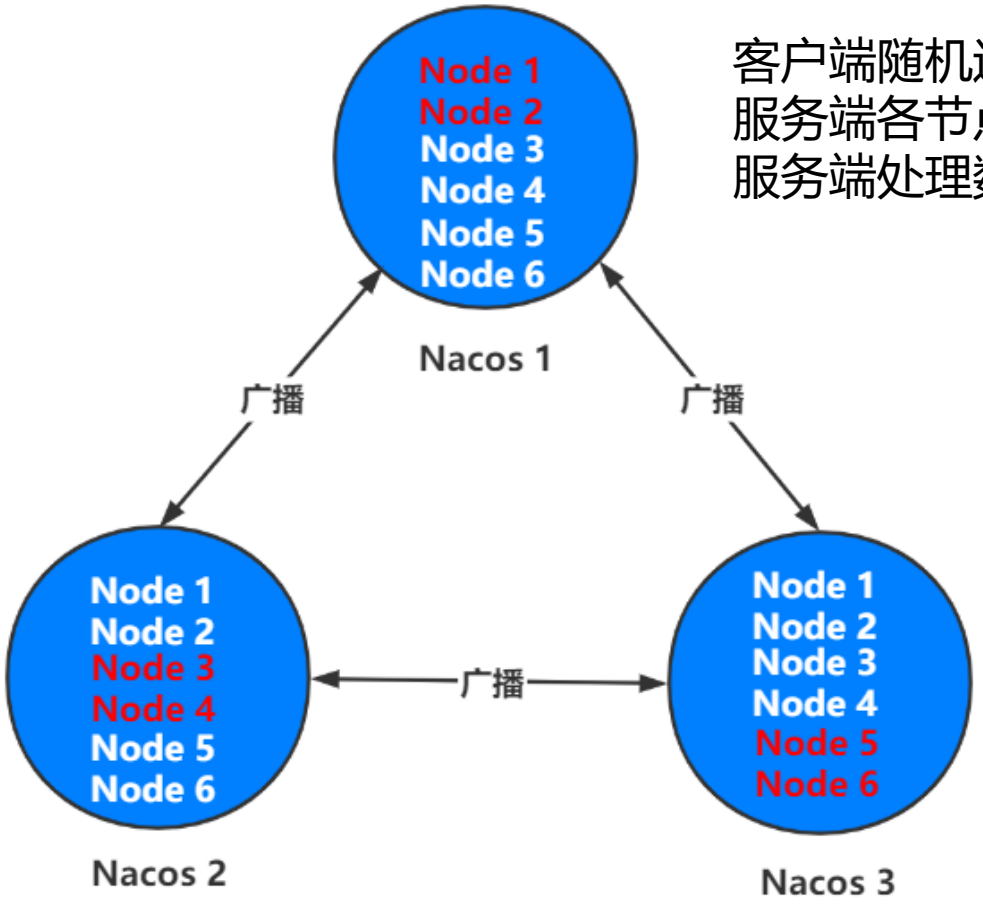
Dubbo 生态系统中的注册中心实现

Nacos实现分析

- 服务注册与健康检查
- 数据模型
- **数据一致性保障**
  - Raft CP一致性
  - **Distro AP一致性**

思考：出现数据不一致如何解决

5秒一次心跳



客户端随机选择节点发送请求
服务端各节点只负责部分数据
服务端处理数据写入，并异步广播通知

01.Nacos注册中心深入分析

奈学教育

Nacos

Dubbo 生态系统中的注册中心实现

Nacos实现分析

- 服务注册与健康检查
- 数据模型
- **数据一致性保障**
  - Raft CP一致性
  - **Distro AP一致性**

异常情况处理:

- 服务端节点宕机
  集群剔除故障节点，重新分配数据
- 集群网络分区
  客户端随机心跳，两集群均为全量数据

02. Sentinel服务治理思路与算法

02.Sentinel 设计原理剖析

奈学教育

Sentinel 介绍

- 定义

  面向云原生微服务的高可用流控防护组件：以流量为切入点，从流量控制、熔断降级、系统自适应保护等多个维度来帮助用户保障微服务的稳定性。

- 资源
  - 可以是应用程序中的任何内容，如URL、服务名
  - 只要通过 Sentinel API 定义的代码，就是资源

- 规则
  - 围绕资源的实时状态设定的规则
    - 流量控制规则
    - 熔断降级规则
    - 系统保护规则
  - 所有规则可以动态实时调整

目的和手段

- 目的
  - 用于对分布式系统中大量微服务进行有效控制管理

- 手段
  - 服务熔断
  - 服务降级
  - 流量控制
  - 系统负载保护
  - 负载均衡

02.服务治理思路与算法

奈学教育

服务熔断

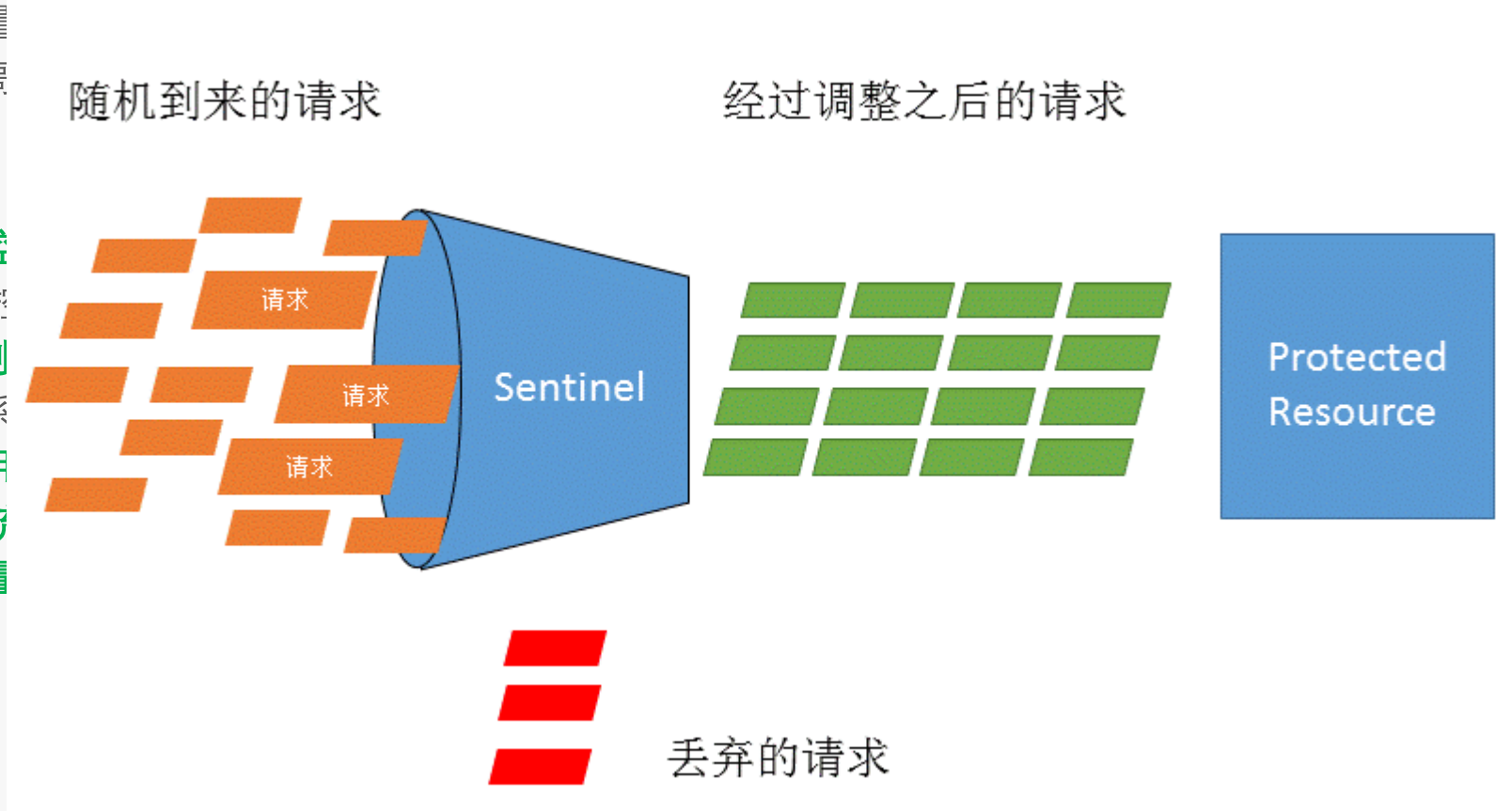
- 定义
  - 在调用服务时，在一些非关键路径服务发生服务质量问题时，尽可能屏蔽问题影响
- 策略
  - 慢调用比例
  - 异常比例
  - 异常数

服务降级

- 定义
  - 服务整体负载超出预设上限阈值或预计流量将超出预设上限时，保证重要或基础的服务能正常运行。
- 策略
  - 服务层降级
    - 拒绝部分请求
    - 关闭部分服务
  - 数据层降级
    - 写MQ，读缓存，异步刷盘
  - 柔行可用策略
    - 自动打开

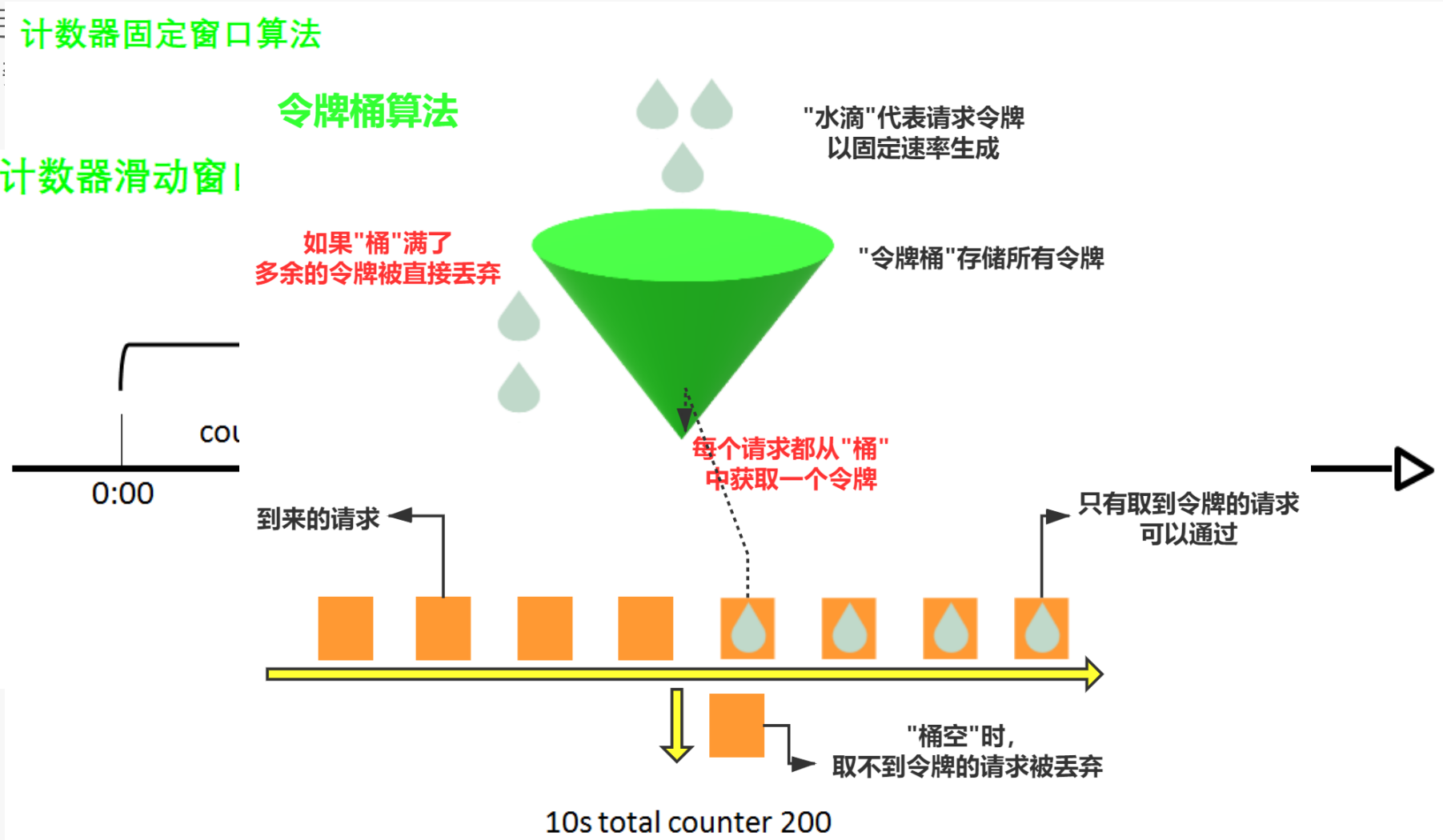
流量控制

- 作用
  - 监控应用流量 流量高峰冲垮，从而保障应用的高
- 手段
  - **并发线程数控**
    - 并发数控
  - **QPS流量控制**
  - 基于调用关系
    - 根据**调用**
    - **链路限流**
    - **关联流量**
- 类型
  - 单机流控
  - **集群流控**
  - 网关流控
  - 系统自适应限流：**在系统不被拖垮的情况下，提高系统的吞吐率，而不是加载这个阈值**



统计算法

- 计数器固定
  - 使用计
- 计数器滑
  - 解决固
- 漏桶算法
  - 以固定
- 令牌桶算
  - 程序以
    牌则通



牌，如获取到令

框架对比

	Sentinel	Hystrix
隔离策略	信号量隔离（并发控制）	线程池隔离/信号量隔离
熔断降级策略	基于慢调用比例、异常比例、异常数	基于异常比例
实时统计实现	滑动窗口（LeapArray）	滑动窗口（基于 RxJava）
动态规则配置	支持多种数据源	支持多种数据源
扩展性	多个扩展点	插件的形式
基于注解的支持	支持	支持
限流	基于 QPS，支持基于调用关系的限流	有限的支持
流量整形	支持预热模式与匀速排队控制效果	不支持
系统自适应保护	支持	不支持
多语言支持	Java/Go/C++	Java
Service Mesh 支持	支持 Envoy/Istio	不支持
控制台	提供开箱即用的控制台，可配置规则、实时监控、机器发现等	简单的监控查看

NX 奈学教育

03.消息队列在分布式架构中的作用

01.消息队列在分布式架构中的作用

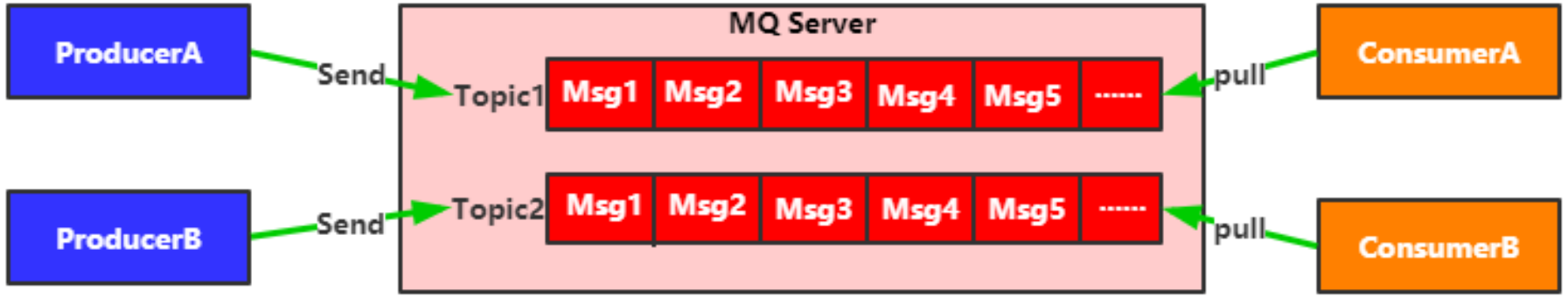
奈学教育

什么是消息队列

消息队列：在消息的传输过程中保存消息的容器，生产者和消费者不直接通讯，依靠队列保证消息的可靠性，避免了系统间的相互影响。

消息队列主要角色

- 服务端
- 客户端
  - 生产者
  - 订阅者



01.消息队列在分布式架构中的作用

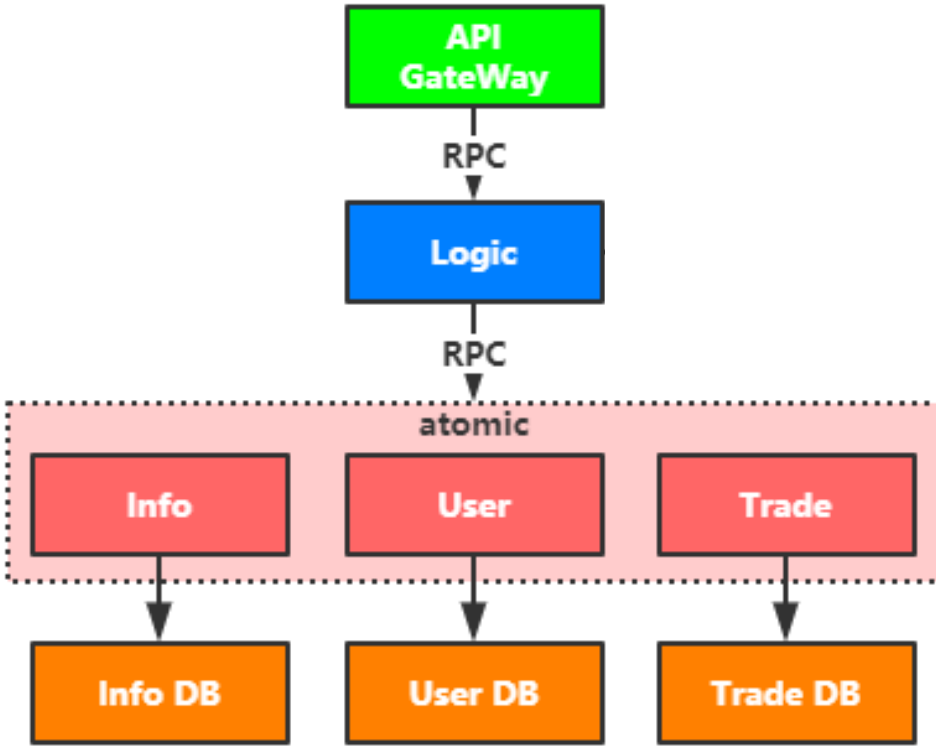
奈学教育

应用场景

运营活动可能需要在业务逻辑中各个环节加入运营活动逻辑，而且有时效性，频繁在正常业务逻辑中添加/删除代码显然不合理且风险极大

运营活动

- 当天交易满3笔送5元红包
- 连续登陆 5天送10元红包
- 新注册用户送5元红包



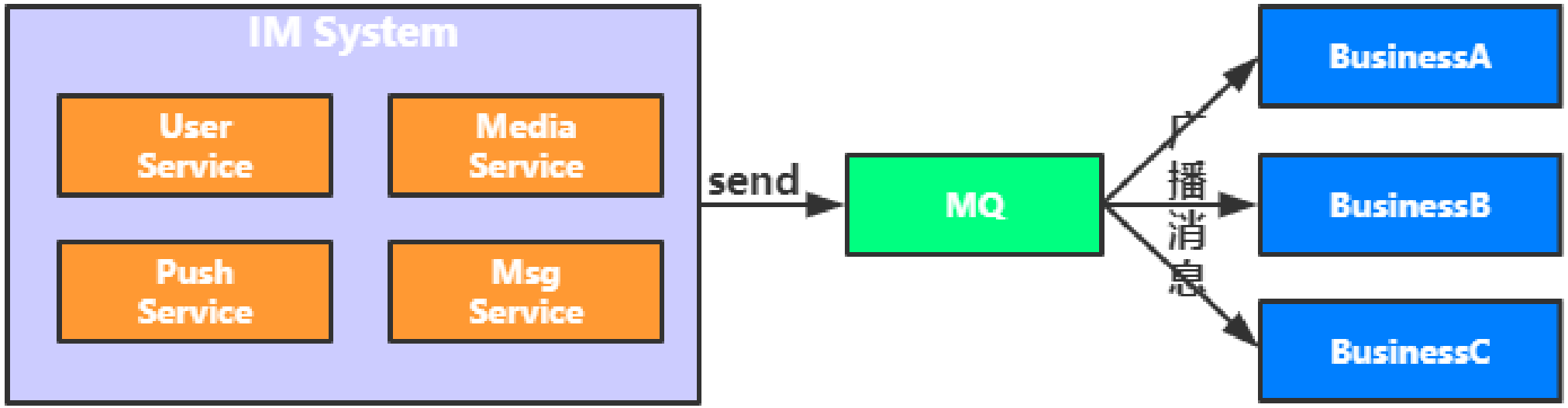
01.消息队列在分布式架构中的作用

应用场景

核心基础服务，可能各个业务线都会关注某些请求处理结果，不断修改代码添加向业务线的通知显然不合理

电商IM系统

- 各业务线对消息接收方自定义逻辑

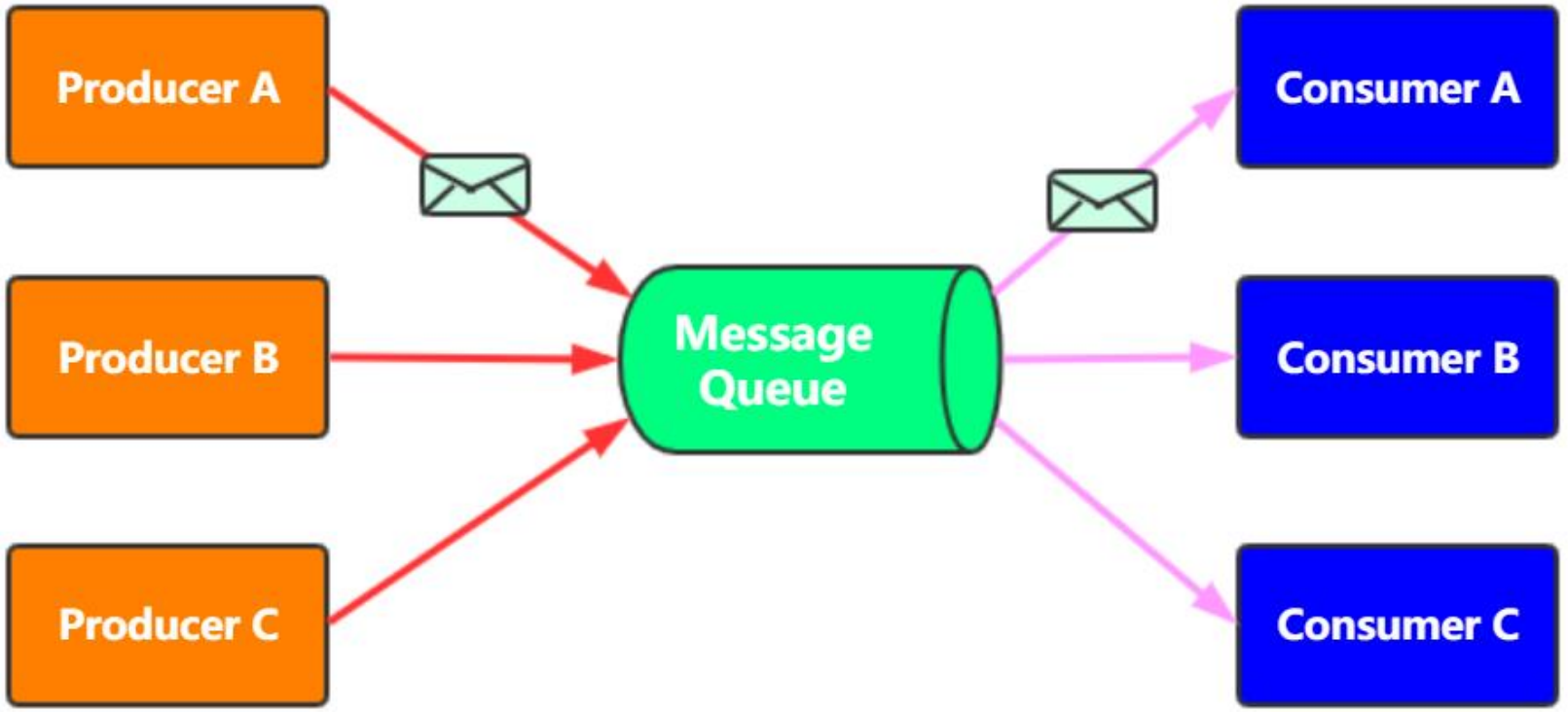


01.消息队列在分布式架构中的作用

奈学教育

消息队列主要作用

- 业务解耦
- 异步调用
- 流量削峰



01.消息队列在分布式架构中的作用

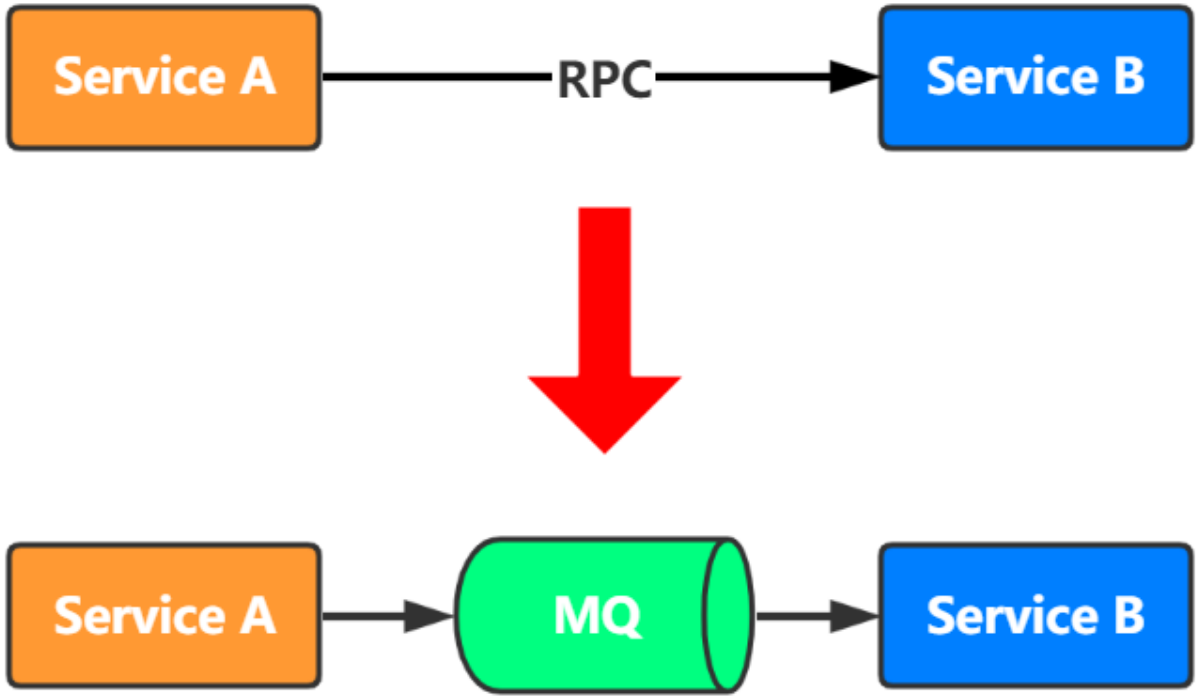
NX 奈学教育

业务解耦

将模块间的RPC调用改为通过消息队列中转，解除系统间的耦合

通过MQ解耦

- 提升系统稳定性
- 通过广播消息避免多次调用
- 提高编码效率



思考：可否完全替代RPC?

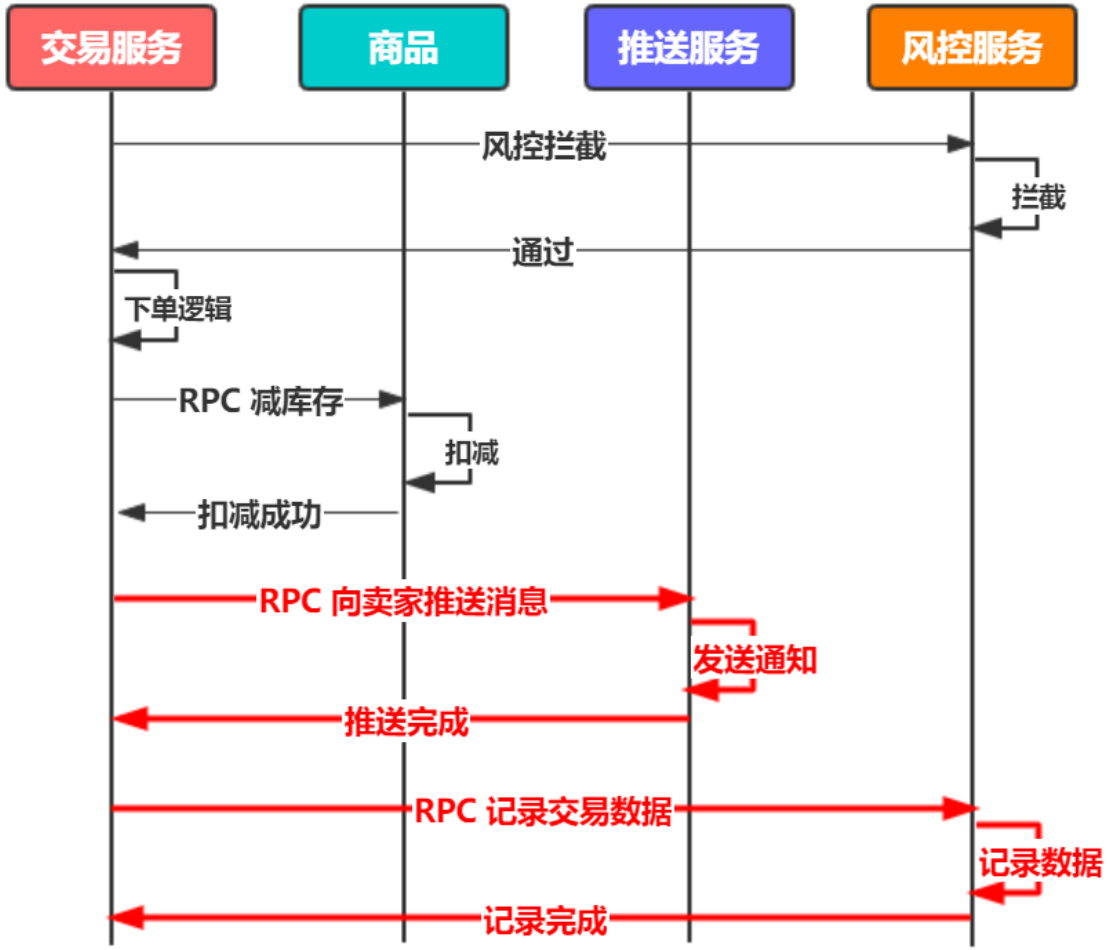
01.消息队列在分布式架构中的作用

异步调用

对于无需关注调用结果的场景，可以通过消息队列异步处理

下单业务处理

- 风控拦截
- 扣减库存
- 通知卖家
- 记录数据



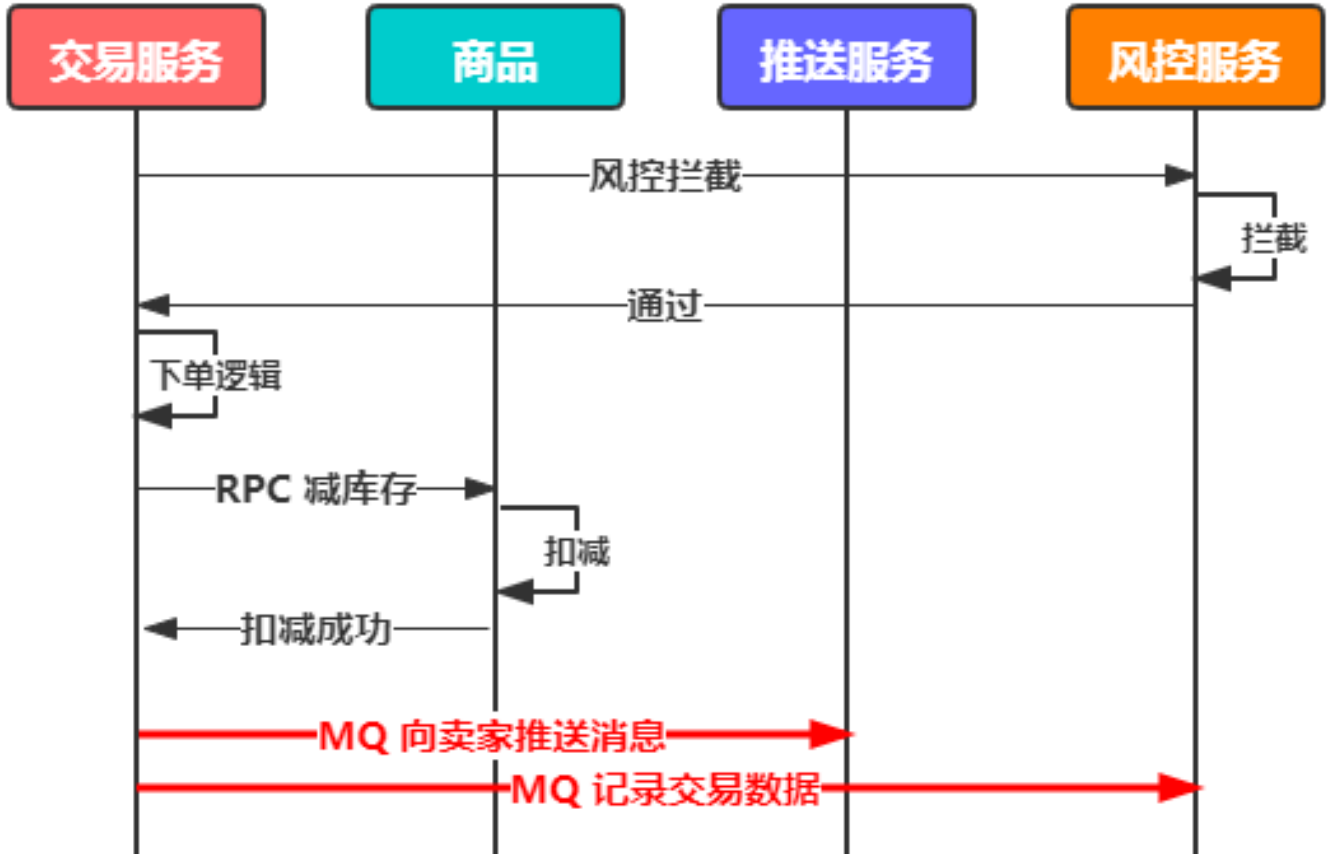
01.消息队列在分布式架构中的作用

异步调用

对于无需关注调用结果的场景，可以通过消息队列异步处理

下单业务优化

- 风控拦截
- 扣减库存
- 异步通知卖家
- 异步记录数据



01.消息队列在分布式架构中的作用

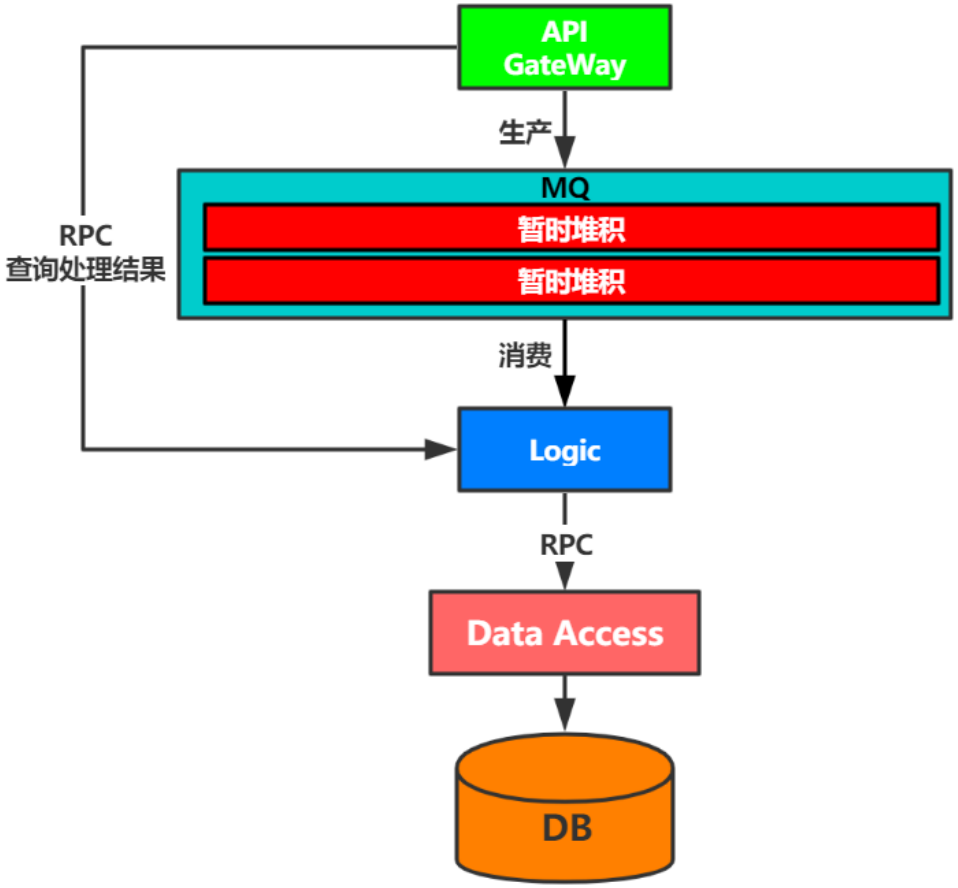
奈学教育

流量削峰

系统的吞吐量往往取决于底层存储服务的处理能力，数据访问层可以调整消费速度缓解存储服务压力，避免短暂的高峰将系统压垮

缓解流量高峰

- 业务逻辑层匀速处理



01.消息队列在分布式架构中的作用

奈学教育

思考

使用消息队列带来很大的收益，但也会对系统架构造成一些负面影响，而且切记不能完全代替RPC，需要合理设计业务调用

对系统架构影响

- 系统可用性
- 架构复杂度
- 排查问题路径

02.主流消息队列选型对比分析

奈学教育

主流消息队列选型

目前主流的MQ主要是RocketMQ、kafka、RabbitMQ，选型是应该从哪几个维度对比

对比维度

- 系统定位
- 支持功能
- 可用性
- 可靠性
- 运维

Apache RocketMQ

APACHE kafka®

RabbitMQ

02.主流消息队列选型对比分析

基础项对比

对比项	kafka	RocketMQ	RabbitMQ
成熟度	日志领域应用广泛	成熟	成熟
公司/社区	Apache	Apache	Mozilla Public License
活跃程度	高	高	高
API完备性	高	高	高
文档完备性	完善	完善	完善
客户端语言	Java、C++、Python、Go	Java	Java、C++、Python

02.主流消息队列选型对比分析

可用性、可靠性对比

对比项	kafka	RocketMQ	RabbitMQ
部署方式	集群	集群	集群
集群管理	Zookeeper	name server	Erlang天然支持
选主方式	自动选举	不支持	最早加入的节点
主从切换	自动切换	不支持自动切换	自动切换
数据可靠性	高	高	高
性能	非常高	高	中
可用性	分布式、主从	分布式、主从	主从
堆积能力	非常好	非常好	一般

02.主流消息队列选型对比分析

功能对比

对比项	kafka	RocketMQ	RabbitMQ
顺序消息	支持	支持	支持
延时消息	不支持	只支持特定Level	不支持
事务消息	不支持	支持	不支持
消息过滤	支持	支持	不支持
消息查询	不支持	支持	不支持
消费失败重试	不支持	支持	支持
批量发送	支持	不支持	不支持

02.主流消息队列选型对比分析

NX 奈学教育

同样是消息队列，差异如此之大?

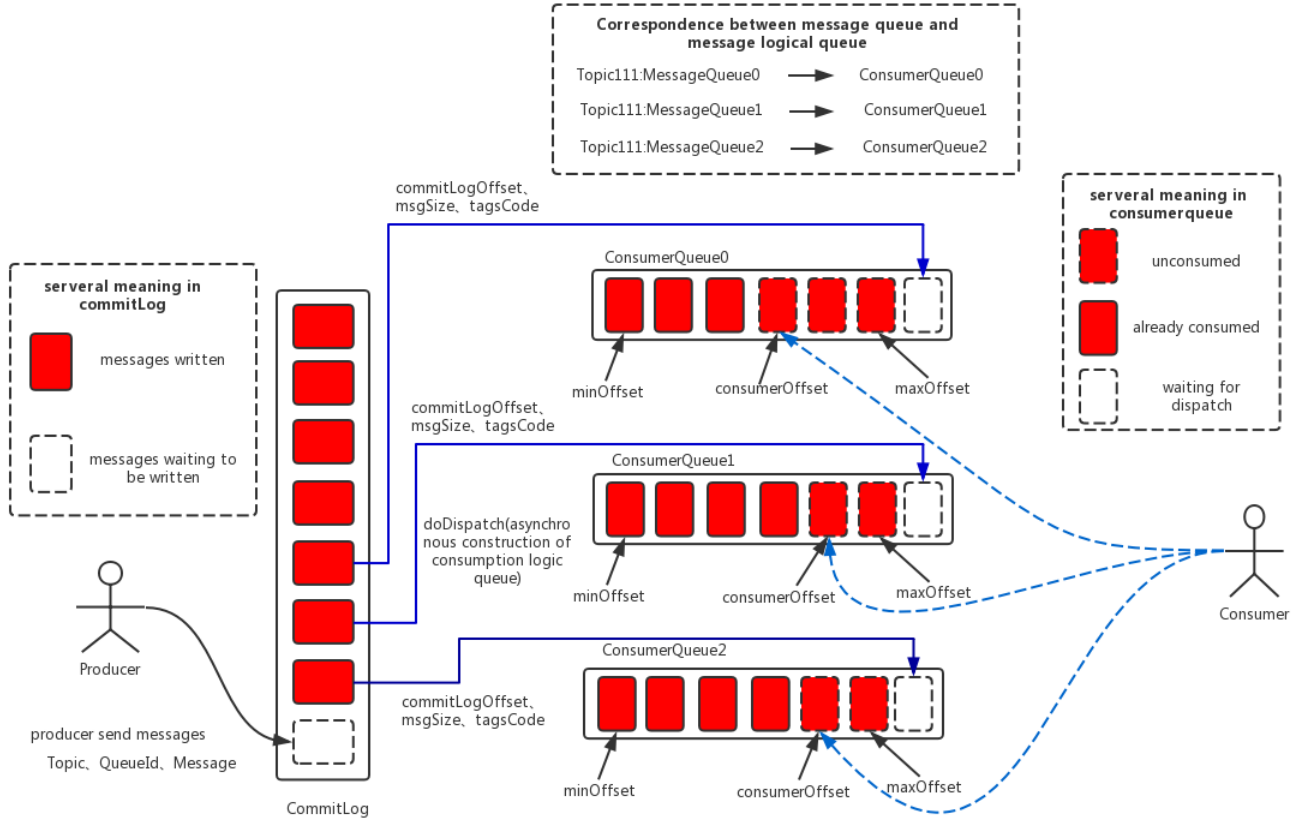
- Kafka：系统间的数据流通道
- RocketMQ：高性能可靠消息传输
- RabbitMQ：可靠消息传输

RocketMQ功能特性

- 支持事务型消息
- 支持延时消息
- 支持消息重发
- 支持consumer端tag过滤
- 支持消息回放

消息存储

- **CommitLog**：存储消息主体
- **ConsumeQueue**：消息消费队列
- **IndexFile**：消息索引文件



顺序写，随机读，如何保证消费性能？？

优化手段

- CommitLog文件切分，默认1G
- MMap提升文件访问性能
- SSD

业务场景分析

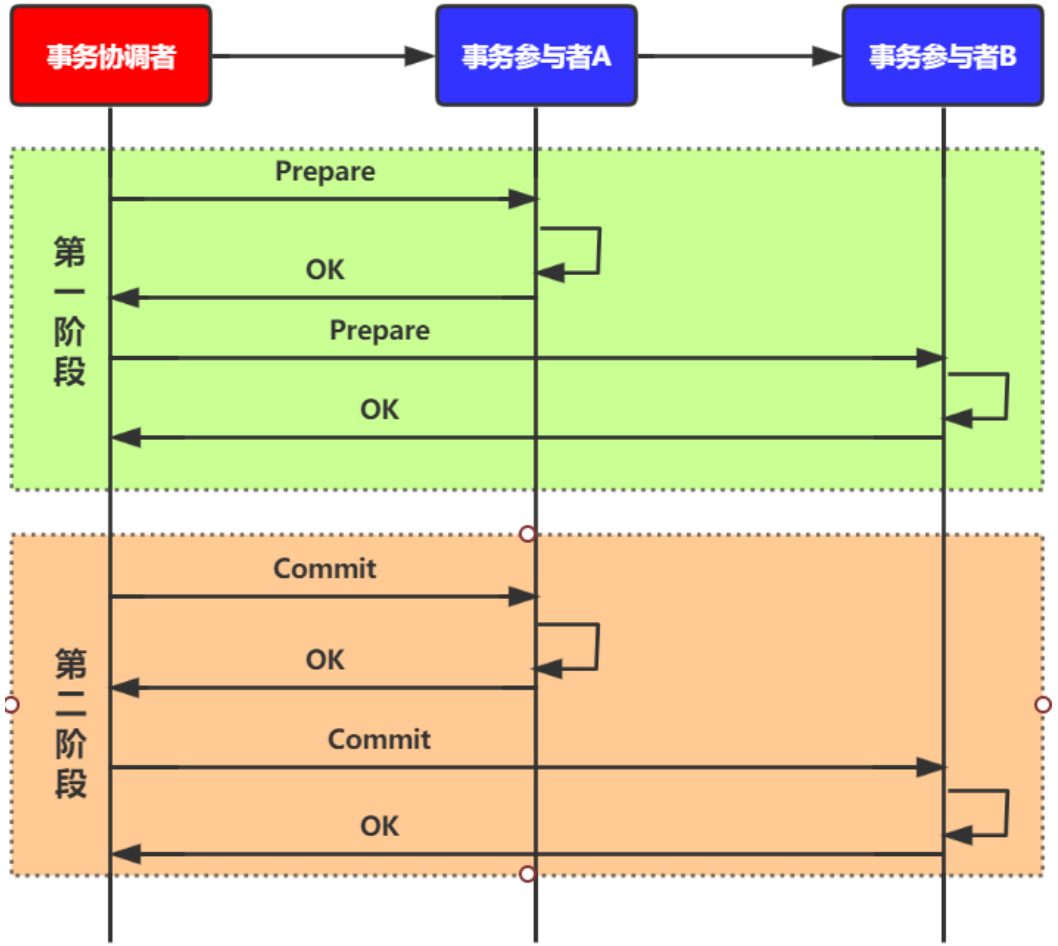
- **用户下单**
  - 创建订单 + 减库存
- **发布或更新商品信息**
  - 写商品库 + 更新外置索引

分布式事务

- 强一致
- 柔性事务
- 事务消息

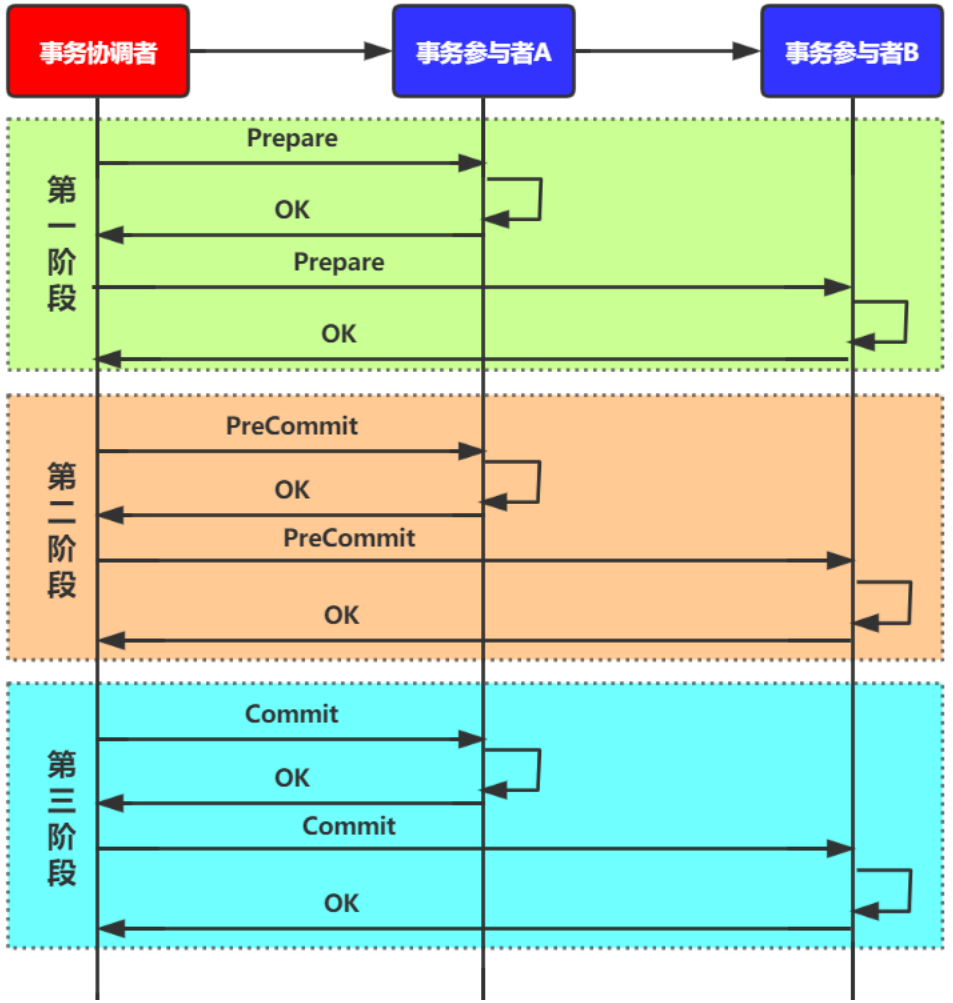
强一致协议

- 一致性协议
  - 两阶段提交 2PC
  - 三阶段提交 3PC
- 落地方案
  - XA规范



强一致协议

- 一致性协议
  - 两阶段提交 2PC
  - 三阶段提交 3PC
- 落地方案
  - XA规范



强一致协议

- 一致性协议
  - 两阶段提交 2PC
  - 三阶段提交 3PC
- 落地方案
  - XA规范
    - 资源管理器-事务参与者
    - 事务管理器-事务协调者

1、写入加锁
2、记录事务执行状态

分布式事务

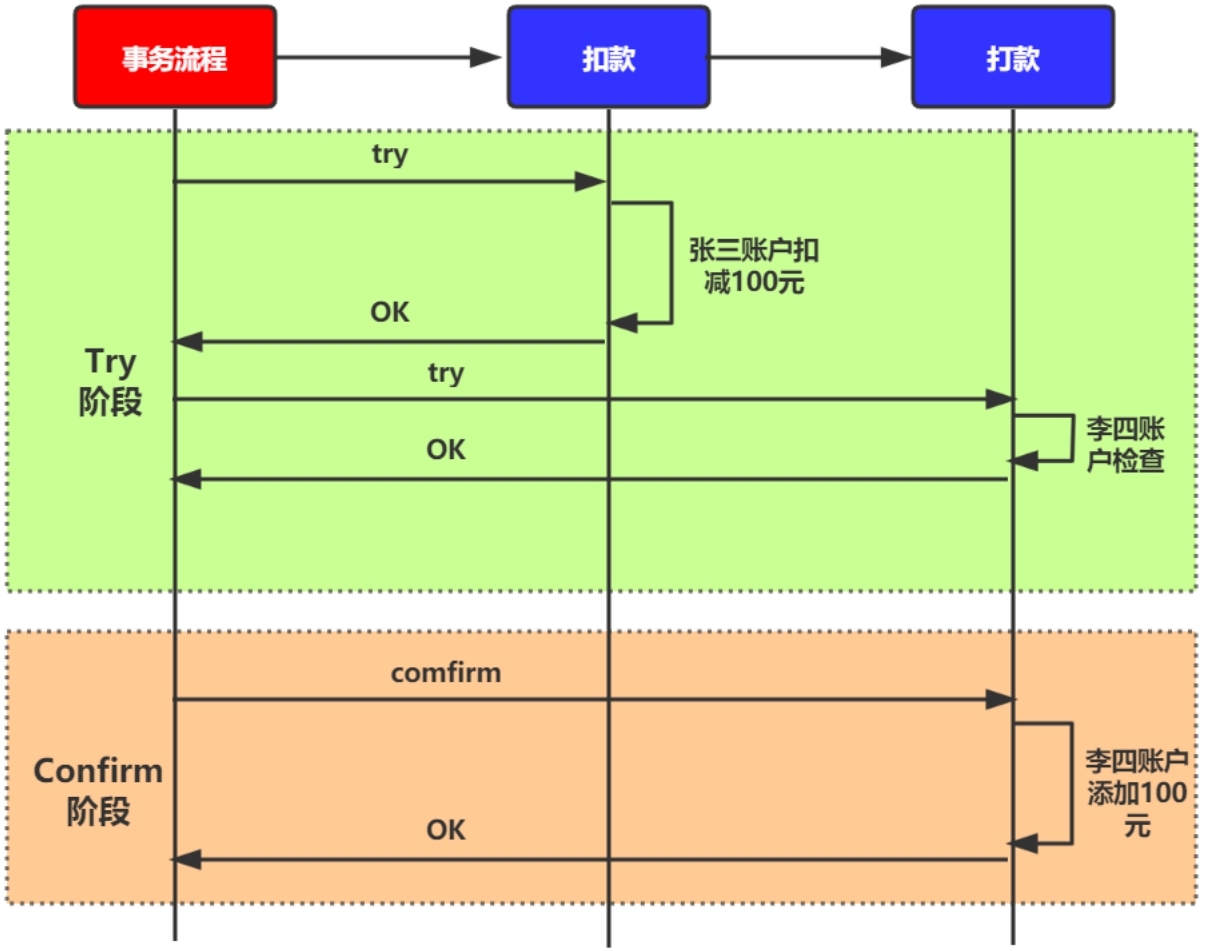
- 强一致
- 柔性事务
- 事务消息

最终一致性方案

- TCC（Try-Confirm-Cancel）
- SAGA模型

最终一致性方案

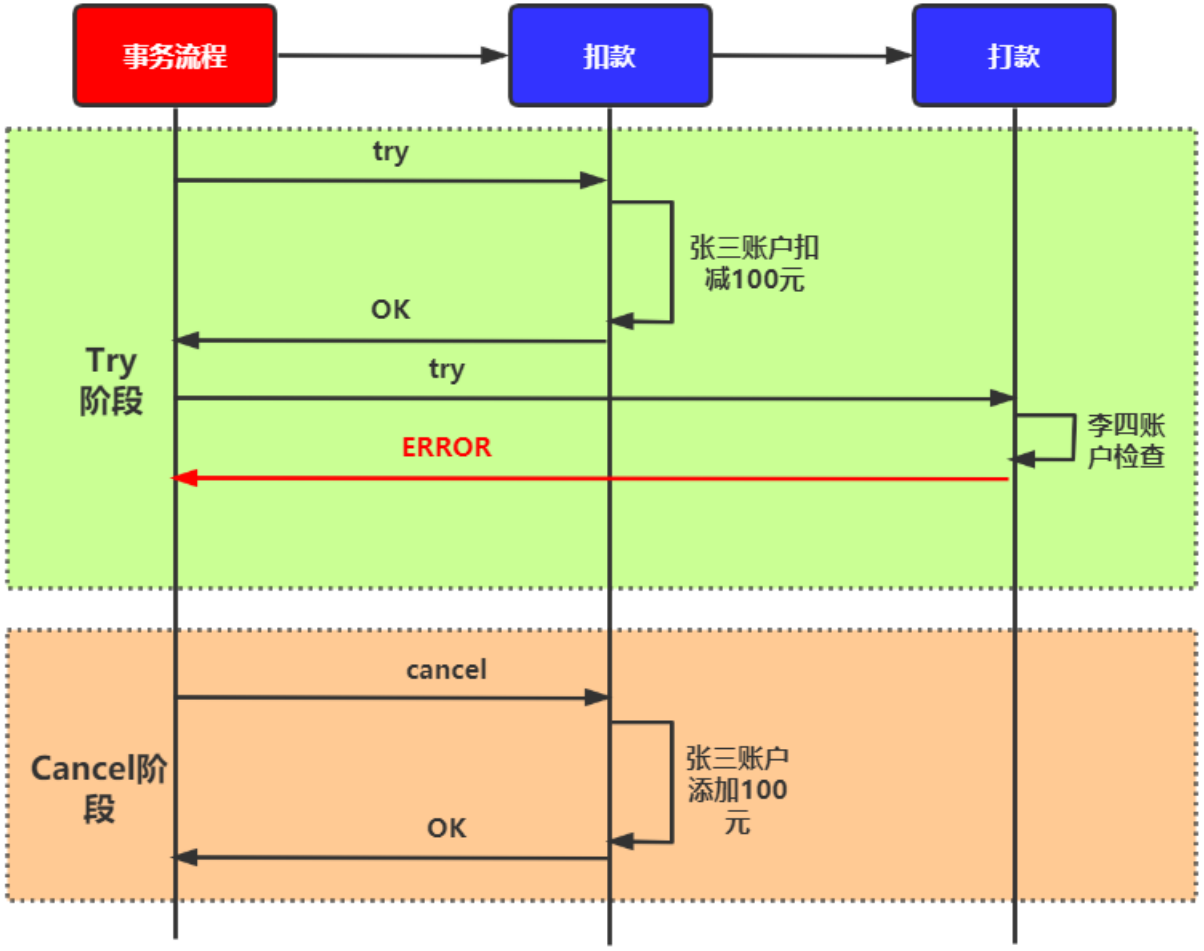
- TCC（Try-Confirm-Cancel）
  - 尝试执行业务，预留资源；
  - 确认执行业务，使用Try阶段资源；
  - 取消执行业务，释放Try阶段预留的资源；
- SAGA模型



TCC成功处理流程

最终一致性方案

- TCC（Try-Confirm-Cancel）
  - 尝试执行业务，预留资源；
  - 确认执行业务，使用Try阶段资源；
  - 取消执行业务，释放Try阶段预留的资源；
- SAGA模型



TCC失败处理流程

最终一致性方案

- TCC（Try-Confirm-Cancel）
- SAGA模型
  - 一个分布式事务拆分为多个本地事务；
  - 本地事务都有相应的执行模块和补偿模块；
  - 事务管理器负责在事务失败时调度执行补偿逻辑；

分布式事务

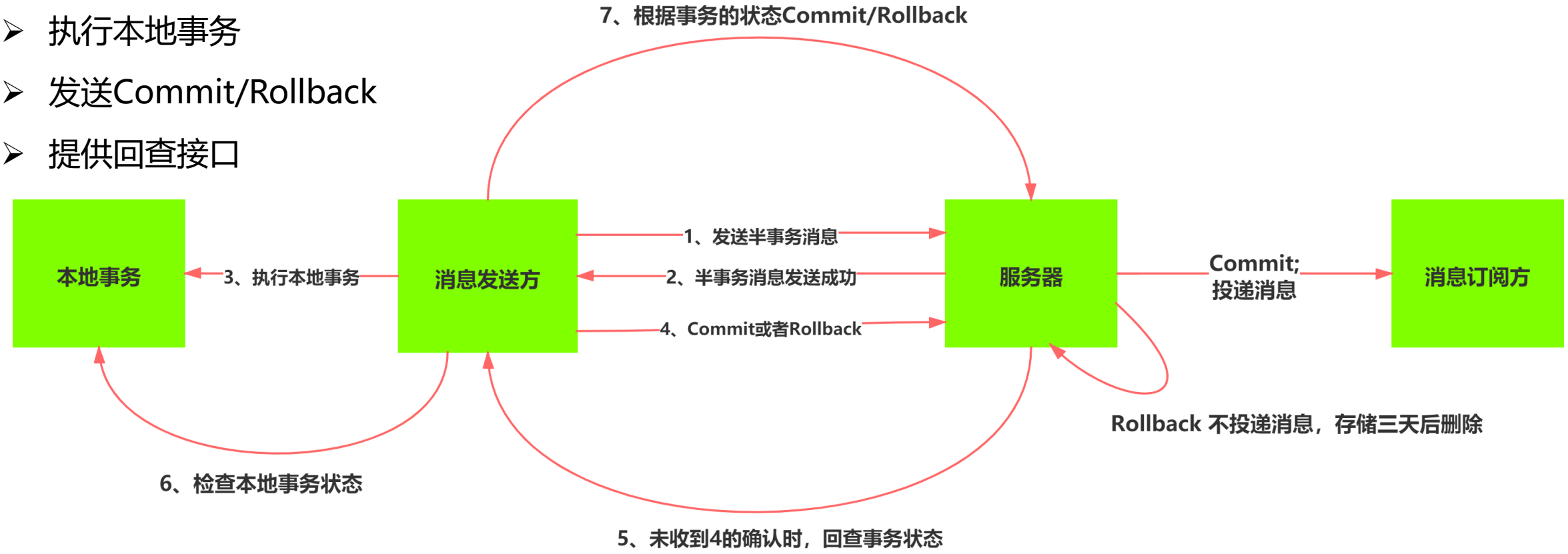
- 强一致
- 柔性事务
- 事务消息
  - 简化了分布式事务模型
  - 对业务友好

两次RPC调用

事务消息+RPC

事务消息实现

- 发送半消息
- 执行本地事务
- 发送Commit/Rollback
- 提供回查接口



RocketMQ事务消息原理

半消息主题

- **HALF消息：RMQ_SYS_TRANS_HALF_TOPIC（临时存放消息信息）**
  - 事务消息替换主题，保存原主题和队列信息
  - 半消息对Consumer不可见，不会被投递
- **OP消息：RMQ_SYS_TRANS_OP_HALF_TOPIC（记录二阶段操作）**
  - Rollback：只做记录
  - Commit：根据备份信息重新构造消息并投递
- **回查：**
  - 对比HALF消息和OP消息进行回查

RocketMQ事务消息原理