Kafka 开发者沟通会

2017-11-30
服务云

爱奇艺
悦 享 品 质

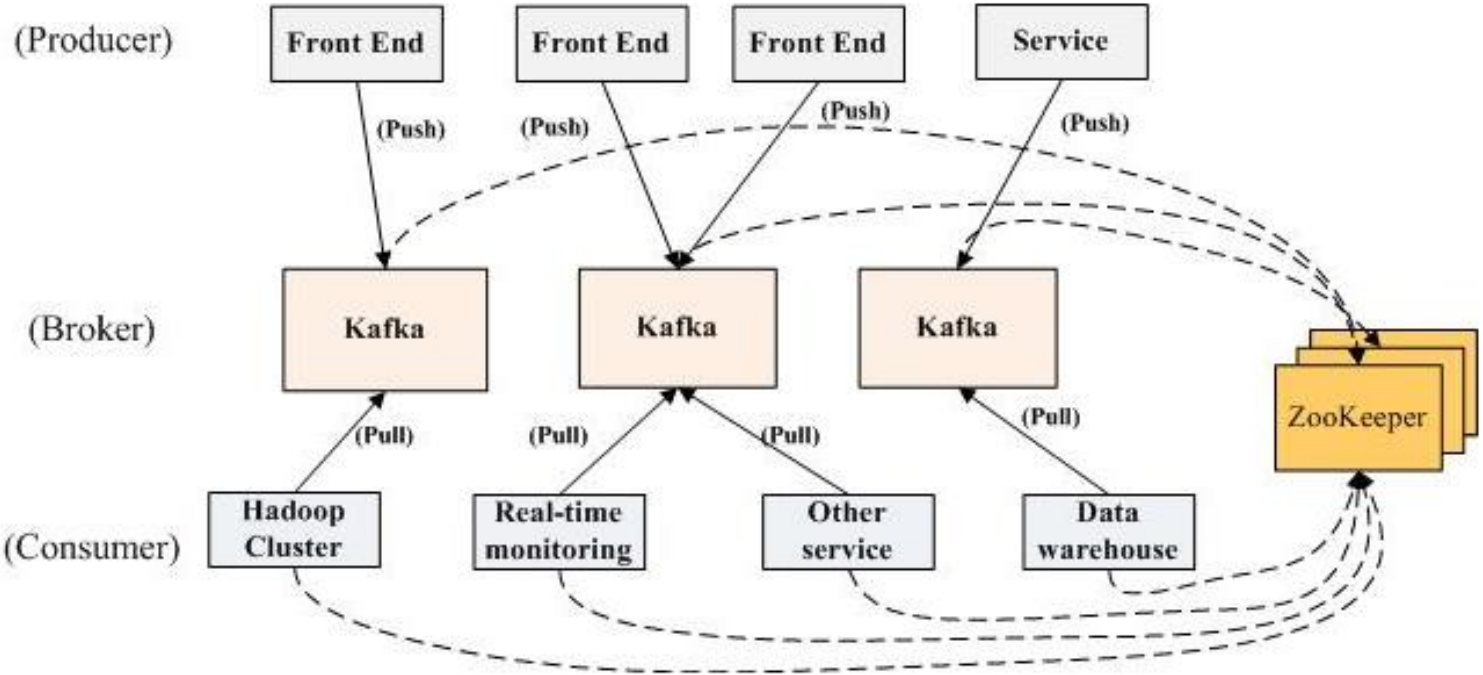
Agenda

- 发展历史
- 架构原理
- 新版本
- 业务场景
- 最佳实践
- 常见问题
- 服务支持

开源历史

- 2010 LinkedIn
- 2011 GitHub
- 2012 Apache/Confluent
- 2013 0.8.x 发布
- 2014 0.9.x 发布
- 2015 0.10.x 发布
- 2017 0.11.x /1.0.x 发布

Kafka架构

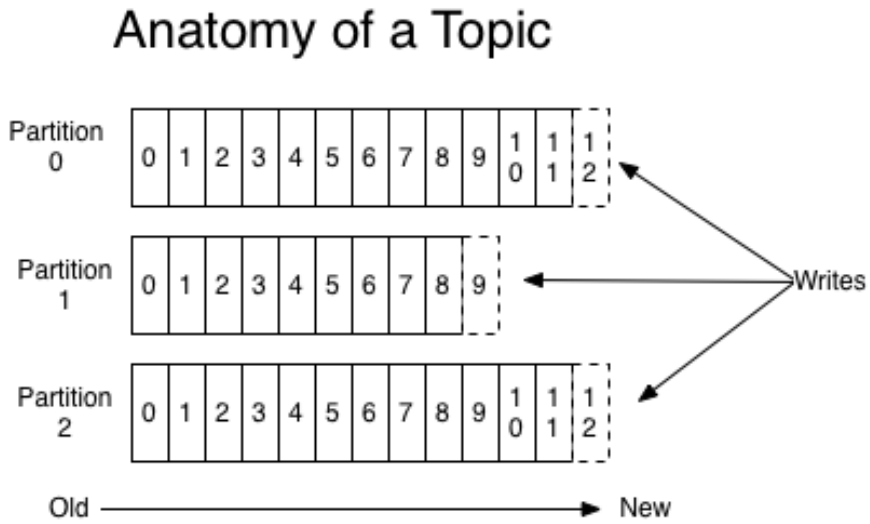


Topic

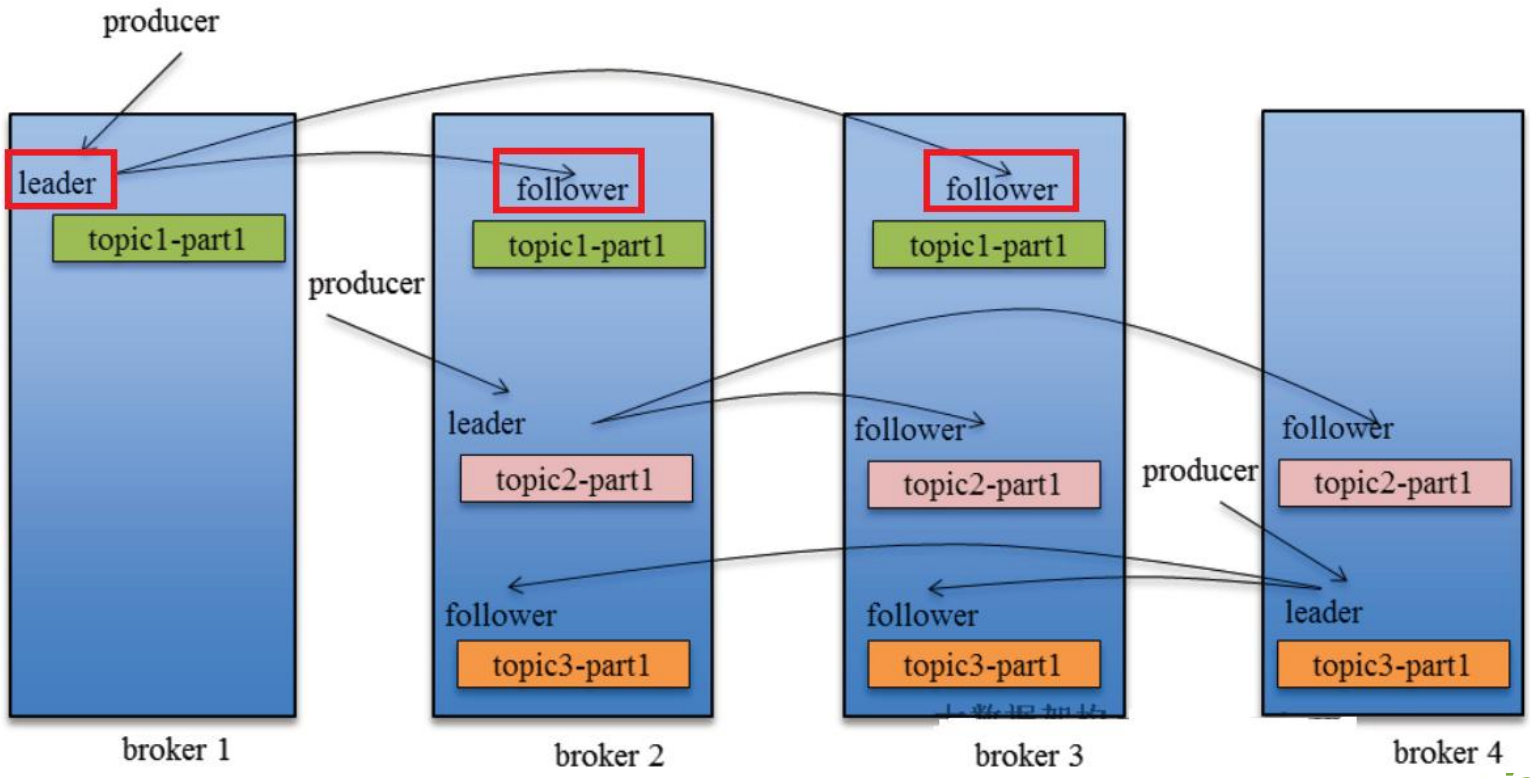
- 逻辑概念，同一Topic的消息分布在一个或多个节点上
- 一个Topic包括一个或多个Partition (多副本，replica)
- 每条消息仅属于一个Topic
- Producer 发布时，必须指定Topic
- Consumer 订阅时，也必须指定订阅的Topic

Partition

- 物理概念，一个partition只分布在一个Broker上(不考虑备份)
- 一个Partition物理上对应一个文件夹
- 一个Partition包括多个Segment(Segment对用户透明)
- 一个Segment 对应一组文件(索引文件、数据文件）
- Segment由多个不变记录组成
- Offset 用来表示一个partition中消息序号



复制原理和同步

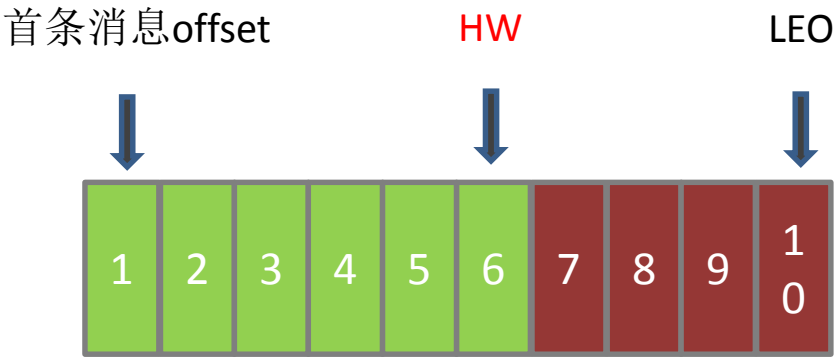


复制原理和同步

- Replica：
  - Topic 的某个partition的副本
  - 对某个partition而言，每个Broker上只会有一个Replica
  - 所有partition的replica默认均匀分配在所有Broker上
  - 高可用保证

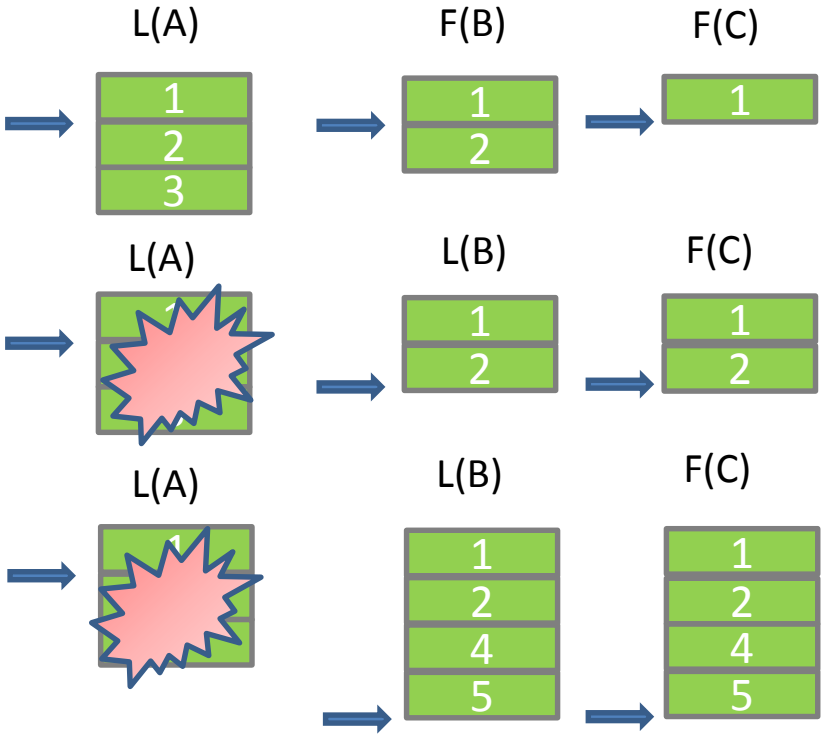
- ISR：
  - Leader会维护一个与其基本一致的Replica列表，称为ISR(in-sync Replica)
  - 如果一个Follower比Leader落后太多，或超过一定时间未收到服务请求，则Leader将其从ISR移除
  - 当ISR所有Replica都向Leader发送ACK时， Leader即Commit .

复制原理和同步



HW: High Water Mark. 一个partition
对应的ISR中最小的LEO作为HW，
consumer最多只能消费到HW所在的位置
LEO: Log End Offset . 一个partition写入的最新消息。

复制原理和同步



1. ISR={A,B,C} Leader A commit m1.

2. A fails, B is new leader. ISR={B,C}
   Leader B commit m2 not m3.

3. B commit m4.m5

复制原理和同步

特例： Replica全部宕机

- 等待ISR中任一Replica恢复，选为Leader
  - 等待时间较长，降低可用性
  - 若ISR中所有的Replica都无法恢复或者数据丢失，则该Partition永不可用

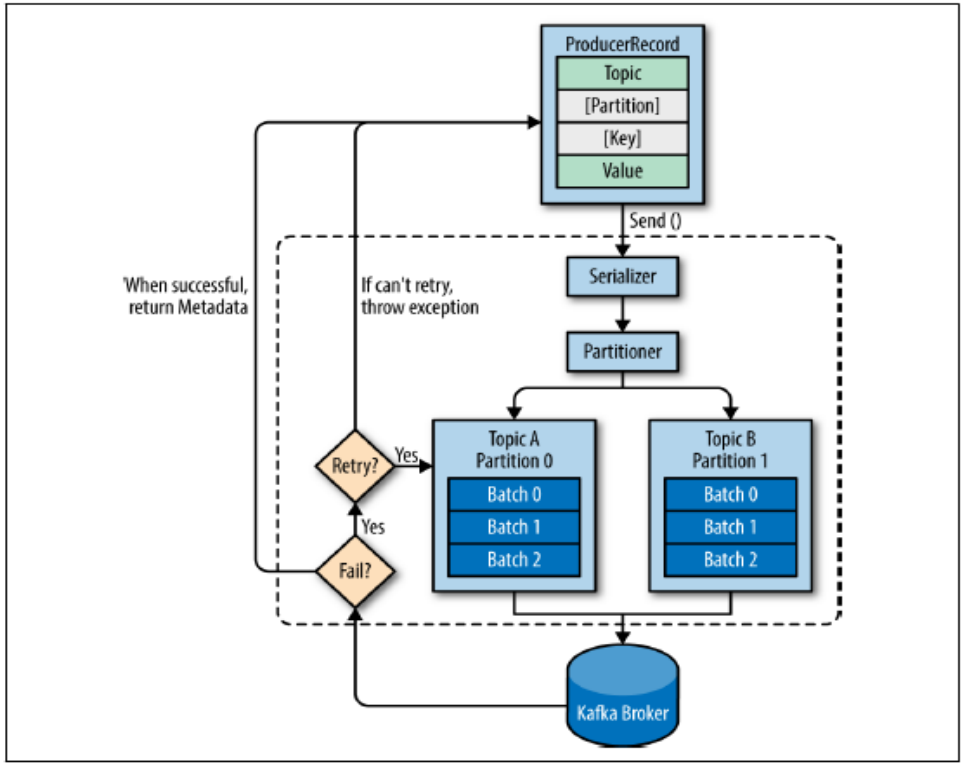
- 选择第一个恢复的Replica为新的leader,无论是否在ISR中 (线上默认)
  - 并未包含所有之前被Leader Commit 过的消息， 因此会数据丢失
  - 可用性较高

Partitioner

- 发送消息，producer通过key来判断该消息落在具体的partition
- 分配策略：
  - ✓ 指定key：
    - Hash：由消息所提供的key来进行hash，然后分发到对应的partition (default)
    - 自定义：自己实现partition接口,配置参数partitioner.class
  - ✓ 未指定key:
    - 随机：把每个消息随机分发到一个partition中。
      在10分钟内，该partition不会切换

建议发送过程中，指定key．有助于消息均匀分布在各个Broker的partition上。

Producer



消息发送过程

同步 vs 异步

- 同步发送
  - 低延时
  - 低吐吞
  - 无数据丢失(需配合ACK)
- 异步发送
  - 高延时
  - 高吞吐率
  - 可能会有数据丢失

acks

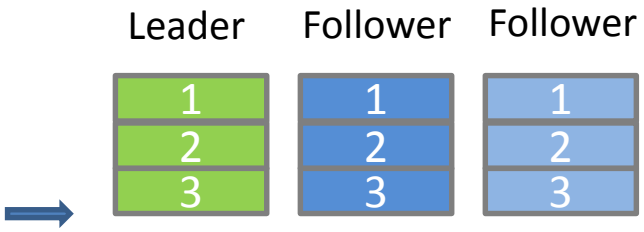
- 1: leader接受成功，则继续发送。
- 0: 无需等待broker 确认，可靠性最低。
- -1: ISR所有follower确认，可靠性最高

可允许集群中一个kafka broker不可用，消息不会丢失：

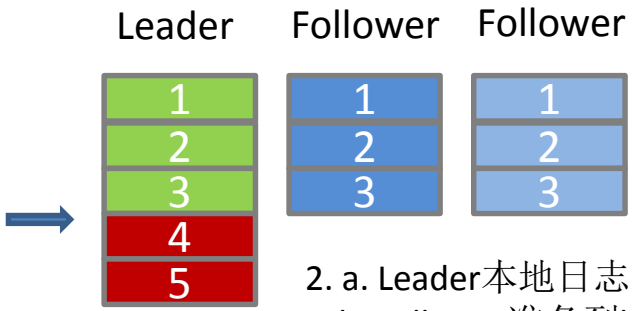
Producer: acks = -1

Topic: min.insync.replicas = 2

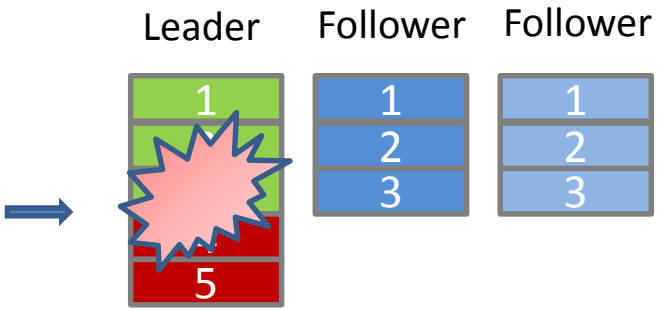
acks = 1



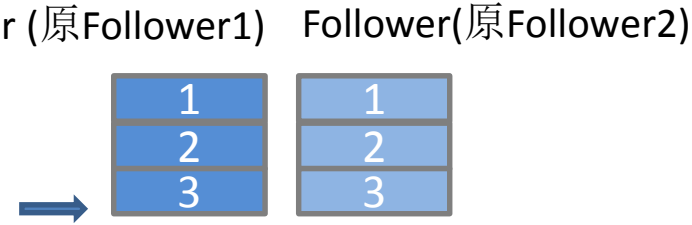
1. Producer发送消息到leader



2. a. Leader本地日志写入成功，返回客户端成功；
   b. Follower准备到leader fetch消息

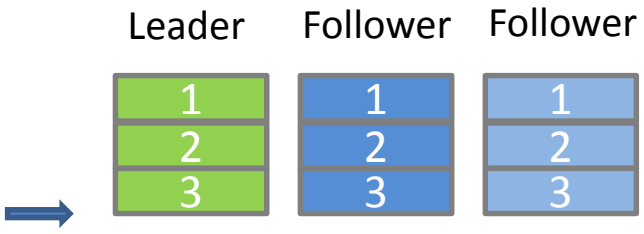


3. a. Follower还没有来得及fetch到最新消息，leader宕机
   b. Follower获取失败，重新选举

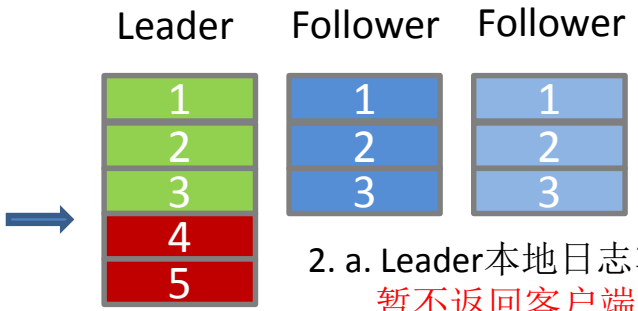


4. Follower1当选为新的leader.
   此时LEO为3， 4,5 消息丢失

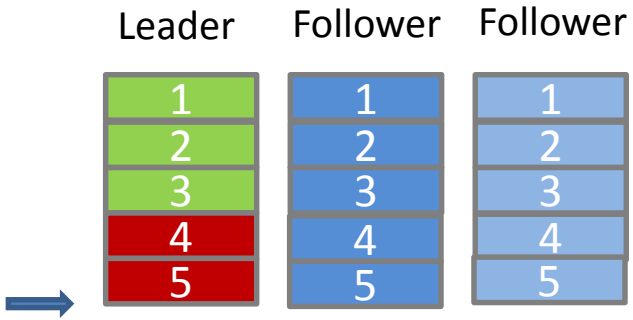
acks = -1



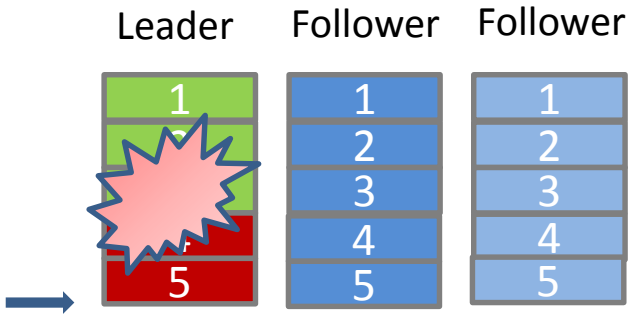
1. Producer发送消息到leader



2. a. Leader本地日志写入成功，
暂不返回客户端成功；
b. Follower准备到leader fetch消息

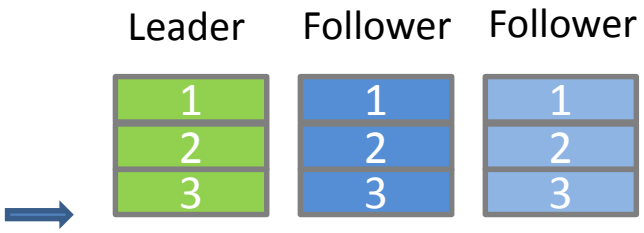


3. a. Follower fetch到最新消息，
leader返回客户端成功

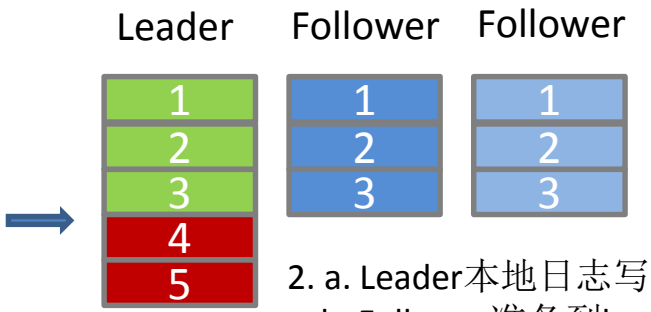


4. Leader 宕机。 从Follower选择新的Leader.
HW 已经到最新LEO, 无数据丢失

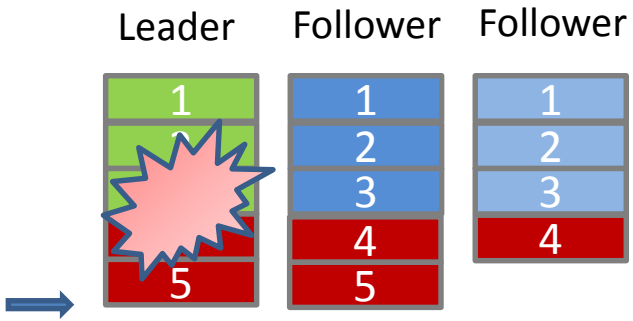
acks = -1



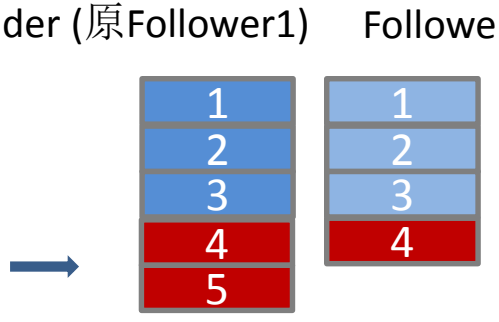
1. Producer发送消息到leader



2. a. Leader本地日志写入成功，暂不返回客户端成功
b. Follower准备到leader fetch消息



3. Follower仍在同步中，follower1 同步完全
Follower2 没有完成。 暂不返回客户端成功5

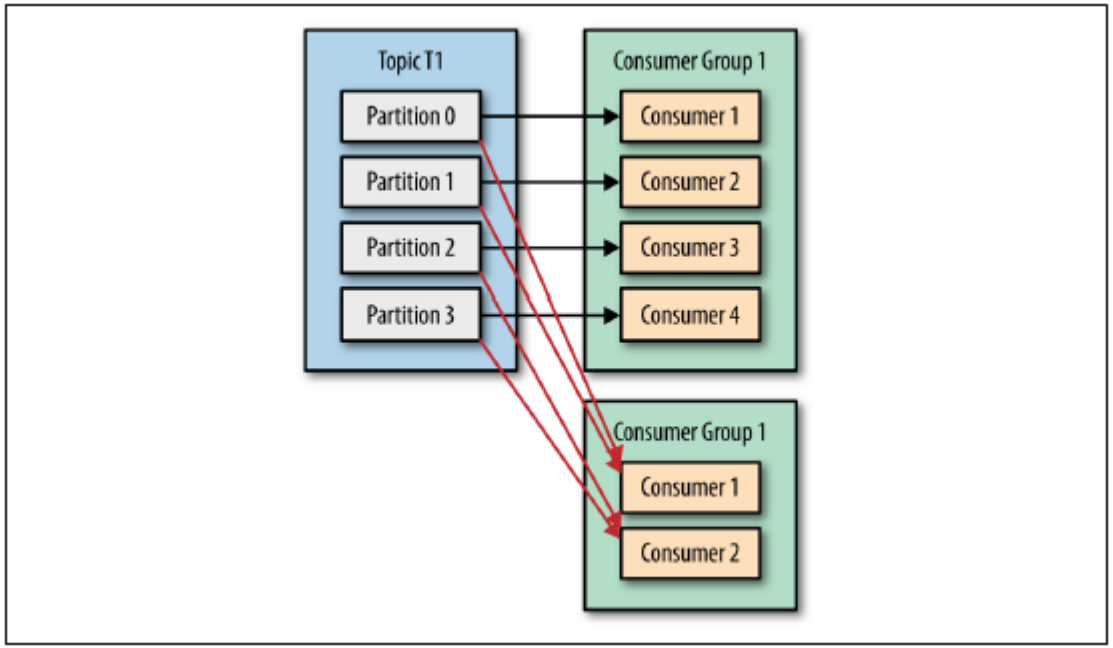


4. a. Follower1 被选为leader, 数据重复5。
b. Follower2 被选为leader , 数据不重复。

High Level Consumer

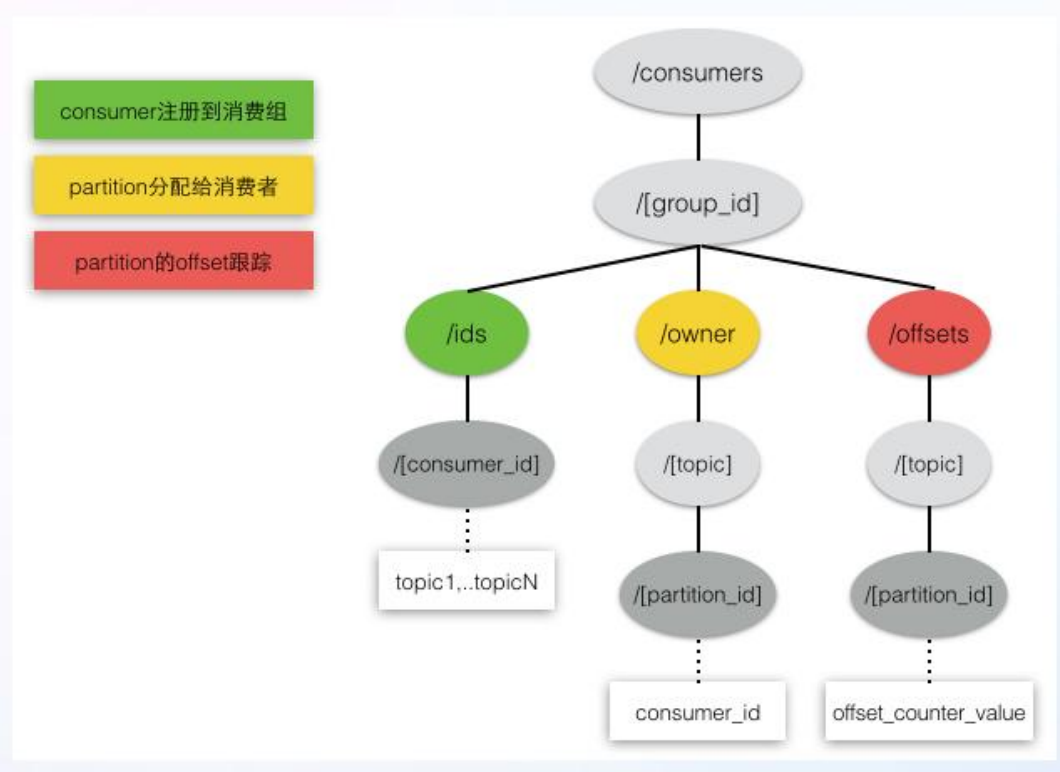
- 消费者主动pull进行拉取消息
- 客户程序只希望从Kafka顺序读取并处理数据，不太关心具体offset.
- 希望提供一些语义，如同一条消息只被某一个Consumer消费（单播）或被所有Consumer消费（广播）
- 提供从Kafka消费的高层抽象，屏蔽细节，并提供丰富语义

Consumer Group



1个partition只能被同一个consumer group中的1个consumer消费，Consumer Num > Partition Num, 会有消费者空闲等待

High Level Consumer



- Consumer Group 的存储结构
- Consumer Lag = Log Size – Consumer offset

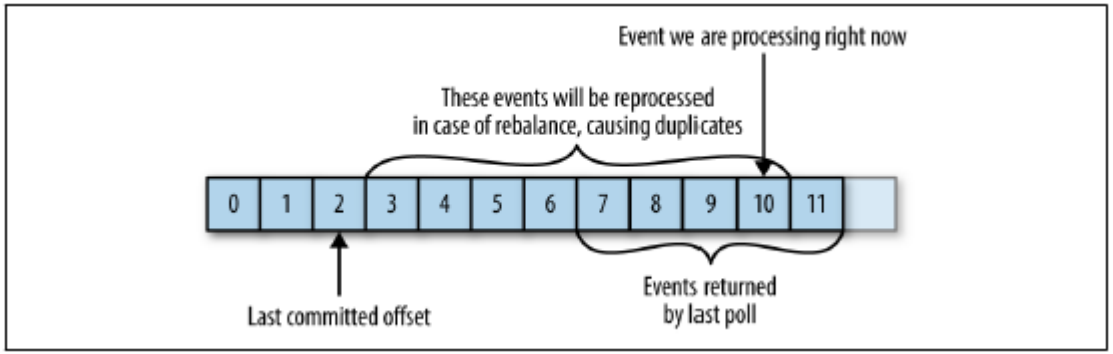
Consumer Group

- CG 集群全局唯一的，而非针对某个Topic
- 消息被消费后，commit offset后，消息也并不会被删除
- 同一个Consumer Group中有消费者数量增加或者减少，或者partition数量增加都会引起 rebalance
- 同一个Consumer Group 中的消费者数量小于等于topic partition 数量
- 消费能力不足时，需要提高partition数量来提高消费并行度

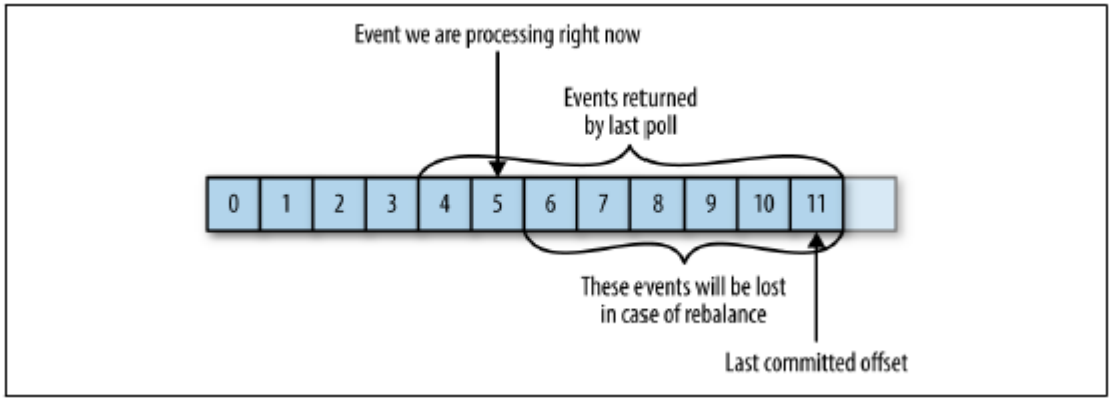
Commit offset

- 自动管理Offset
  - auto.commit.enable=true
  - auto.commit.interval.ms=60*1000
- 手工管理offset
  - ConsumerConnector.commitOffsets();

Commit offset



- 此种情况会重复处理消息



- 此情况会错失处理部分信息

Low Level Consumer

- 更方便管理消费
  - 多次消费
  - 指定partition进行消费，不动态分配partition给消费者

- 额外工作量
  - 跟踪处理offset
  - 获取Partition leader, leader变化时要进行更新
  - 处理多Consumer 协作

New API

- Old API : 0.8.x
- New API : Since 0.9.x
- 兼容性 :

	Producer		Consumer	
Server	0.8	0.10	0.8	0.10
0.8	兼容	N/A	兼容	N/A
0.10	不兼容	兼容	兼容(部分)	兼容

建议使用Server配套的新API .

New API

- Producer
  - 增加异步发送回调
  - 重构Partitioner接口
- Consumer
  - subscribe 动态分配 vs assign 手工指定partition，两者互斥
  - 也有Consumer Group概念
  - 可对Consumer Rebalance进行监听
  - 手动commit offset,也可以本地保存offset
  - 控制消费位置，即从指定offset开始消费
  - 可根据时间戳获取offset

Kafka Streams

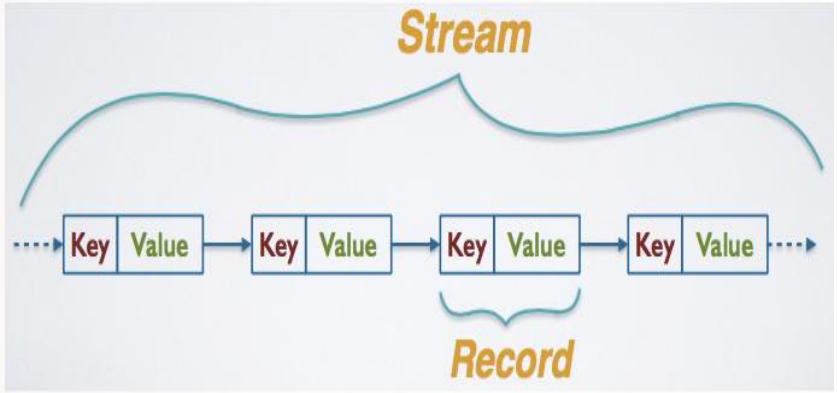
- **Powerful:** 功能强大
  - 高可用、可扩展性、故障容错

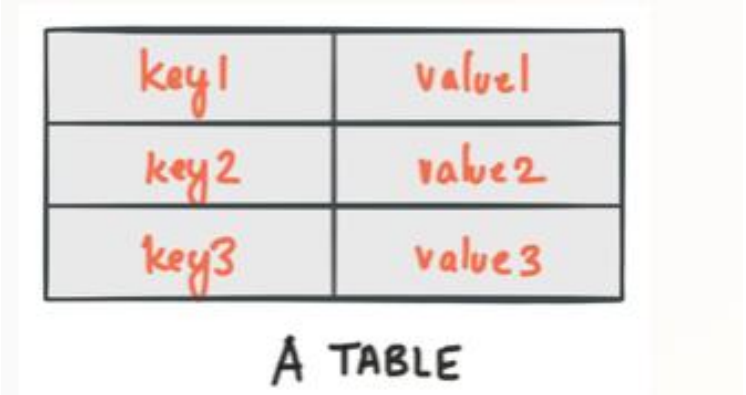
- **Lightweight: 轻量级**
  - 不需要专用集群
  - 不需要外部依赖
  - 它是一个客户端库，不是一个框架

- **Fully integrated:** 完全完整的
  - Kafka 0.10完全兼容
  - 和已有应用程序容易集成
  - 对部署方式没有严格的规则限制

- **Real-time:**实时的
  - 微秒级别的处理延迟
  - 不是micro-batch处理

KStream & KTable

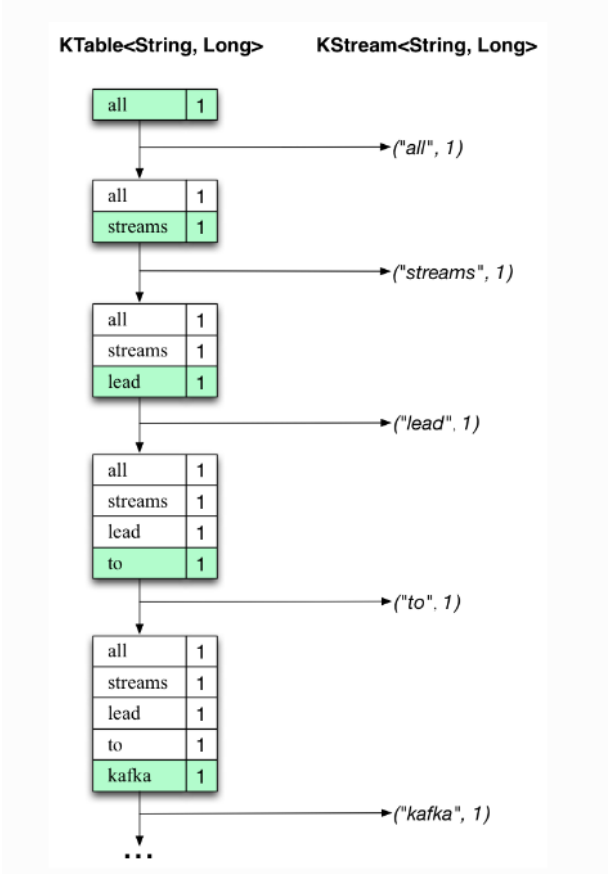


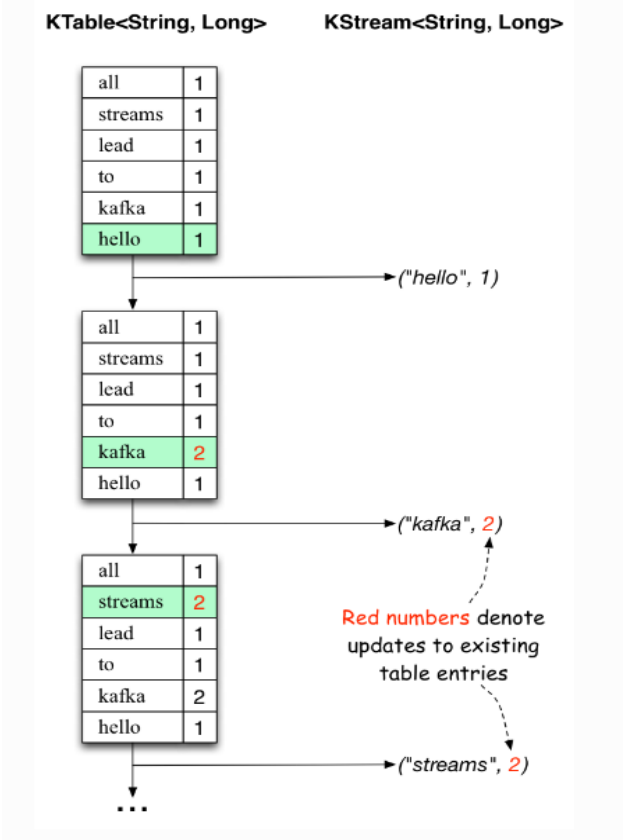


一个KStream是对记录流的抽象，每条数据记录能够表示在无限数据集中自包含的数据。记录流中的数据只有追加，不会有记录会替换已有的相同key的行。比如信用卡交易、访问时间、服务端日志条目；

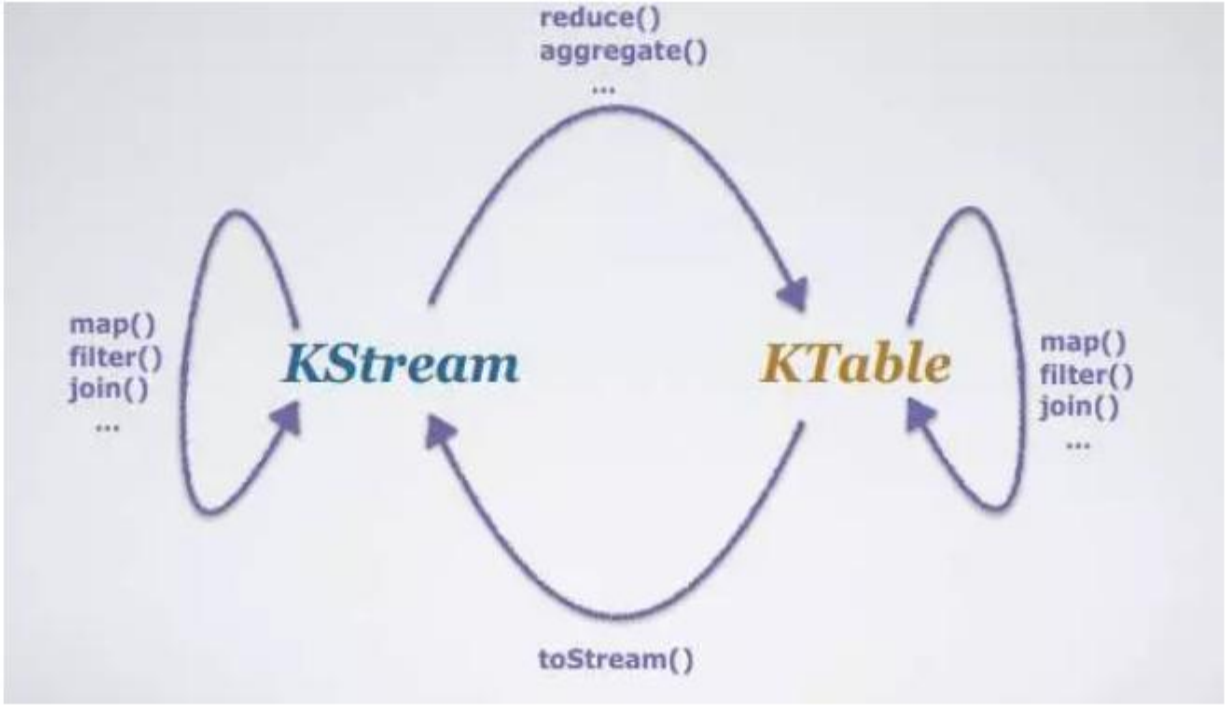
一个KTable是对变更日志流的抽象，每条数据记录代表的是一个更新。如果存在key则更新，如果key不存在，更新操作会被认为是创建。不存在相同的key；

KStream & KTable





KStream & KTable



http://wiki.qiyi.domain/pages/viewpage.action?spaceKey=~luxiaoshuang&title=Java+8%3A+Stream

可靠性保证

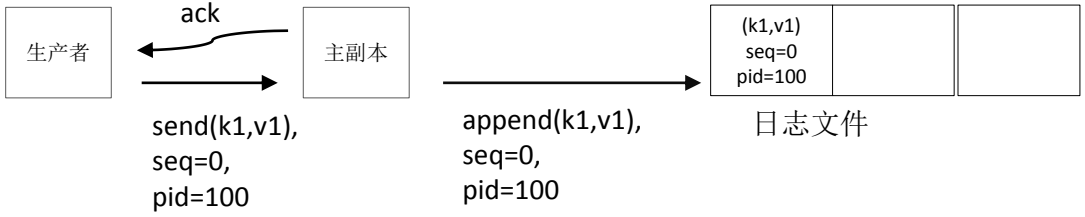
- At least once: 消息绝不会丢，但可能会重复传输
- At most once: 消息可能会丢，但绝不会重复传输
- Exactly once：每条消息肯定会被传输一次且仅传输一次

Kafka 保证的是At least once .

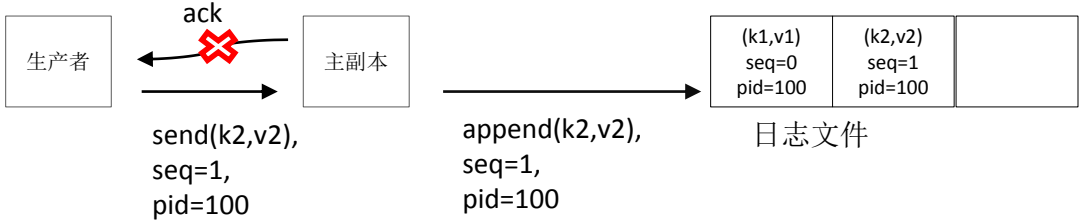
Exactly Once

- 版本： 0.11, 1.0
- 原理：
  - 消息添加序号（seq） + 生产者编号（producerID）写入日志文件
  - 判断是否重复，若重复拒绝写入
- 限制
  - 同一分区的消息不重复，如果多个分区不保证
  - 只保证一个生产者级别幂等；如果生产者多个或不同生产者不保证

Exactly Once



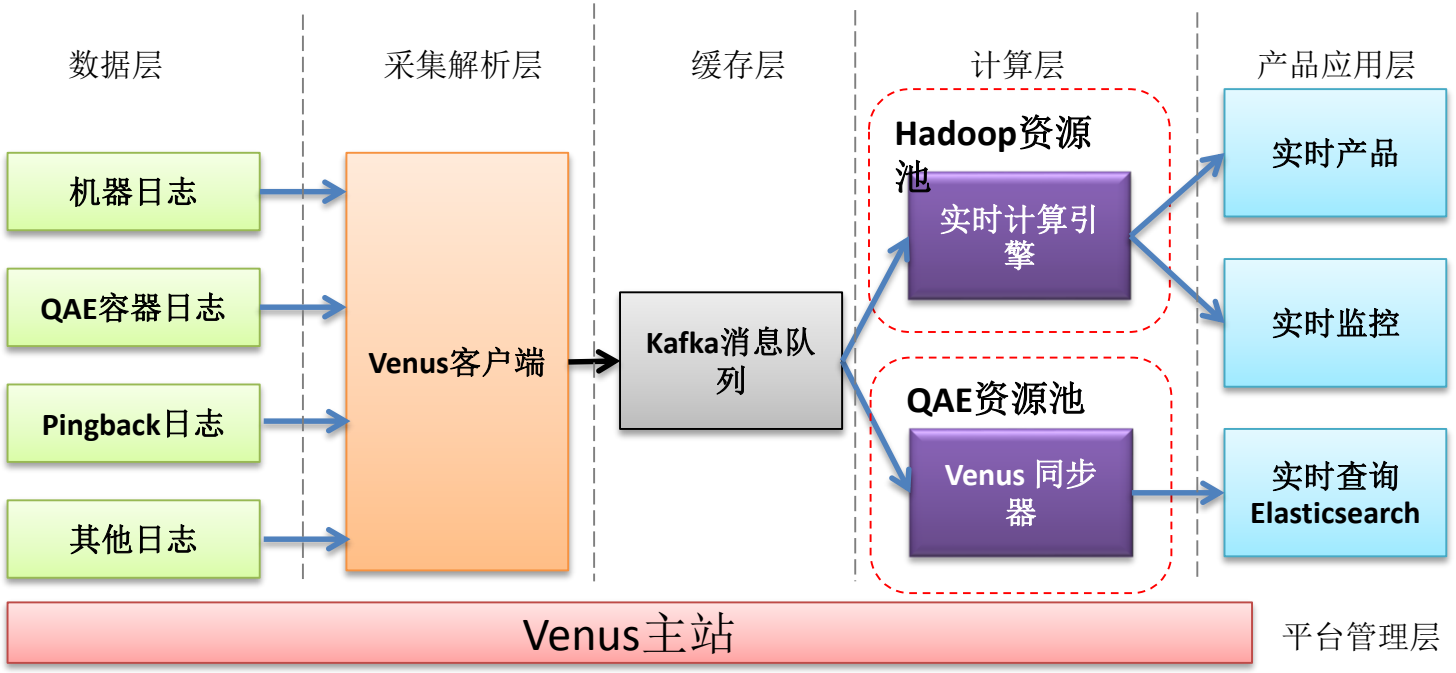
发送消息（K1,V1）



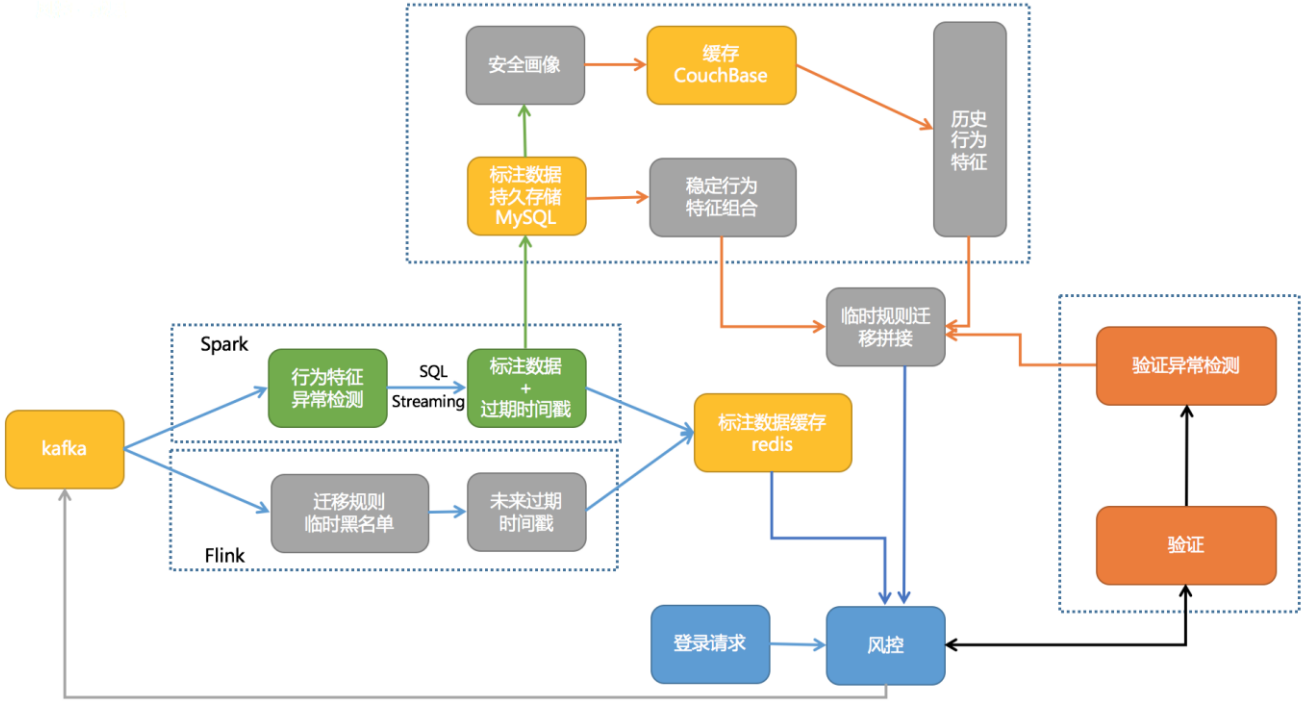
发送消息（K2,V2），
消息写入成功， ACK丢失

重发（K2,V2),
发现消息重复

场景1-Venus (离线)



场景2-安全风控（实时）



最佳实践(生产者)

- 使用新版本API
- 消息发送失败的重试应该使用Producer本身的重试机制
- 保证数据均匀的分布在Topic的Partition中，设置key不为null,且不相同
- 批量发送
- 控制单条消息体大小，勿超限
- 合理配置acks参数项来保证消息不丢失

最佳实践(消费者)

- 使用新版本API
- 消费不同的topic,设置不同的group.id,避免rebalance相互影响(对High level consumer与New consumer)
- 保证fetch.message.max.bytes(client) > message.max.bytes(server)的值
- 消费线程数小于或者等于partition,保证消费者不空闲
- rebalance.max.retries * rebalance.backoff.ms > zookeeper.session.timeout.ms
- 关注Consumer Group 的Consumer Lag (仅对High level consumer)
- 异步发送，注册发送callback关注发送状态 (new produce API)
- 设置合理partition数量

常见问题

- Kafka 是否会丢数据？是否会消息重复？
- Kafka对顺序是否有保证？
- Kafka 是否支持生产、消费的权限控制？如何控制？
- Kafka消息可以不删除吗？
- 想使用Kafka Stream， 该怎么使用？
- 提供高可用方案吗？

常见问题(代码异常)

Produce:

- MessageSizeTooLarge
- java.lang.ClassCastException: java.lang.String cannot be cast to [B
- NotEnoughReplicas
- NotLeaderForPartitionException

Consume:

- NotLeaderForPartitionException
- InvalidMessageSizeException
- ConsumerRebalanceFailedException
- Offset out of range

服务支持

- 开发者手册：http://doc.gitlab.qiyi.domain/Kafka/
- 邮件组：
  - cloud-kafka@dev.qiyi.com
- 对接人：
  - ✓何晓娟（hexiaojuan@qiyi.com、13661636518）
  - ✓冯浩（fenghao@qiyi.com、13816903910）

爱奇艺

悦　　享　　品　　质