

SOFAStack

消息队列 SOFAKafka 技术白皮书

产品版本：AntStack Plus 1.13.1


文档版本：20230707

法律声明

蚂蚁集团版权所有©2022，并保留一切权利。

未经蚂蚁集团事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

商标声明

 蚂蚁集团
ANT GROUP 及其他蚂蚁集团相关的商标均为蚂蚁集团所有。本文档涉及的第三方的注册商标，依法由权利人所有。

免责声明

由于产品版本升级、调整或其他原因，本文档内容有可能变更。蚂蚁集团保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在蚂蚁集团授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过蚂蚁集团授权渠道下载、获取最新版的用户文档。如因文档使用不当造成的直接或间接损失，本公司不承担任何责任。

通用约定

格式	说明	样例
 危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 危险 重置操作将丢失用户配置数据。
 警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	 警告 重启操作将导致业务中断，恢复业务时间约十分钟。
 注意	用于警示信息、补充说明等，是用户必须了解的内容。	 注意 权重设置为0，该服务器不会再接受新请求。
 说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击设置> 网络> 设置网络类型。
粗体	表示按键、菜单、页面名称等UI元素。	在结果确认页面，单击确定。
Courier字体	命令或代码。	执行 <code>cd /d C:/window</code> 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	<code>bae log list --instanceid</code> <i>Instance_ID</i>
[] 或者 [a b]	表示可选项，至多选择一个。	<code>ipconfig [-all -t]</code>
{ } 或者 {a b}	表示必选项，至多选择一个。	<code>switch {active stand}</code>

目录

1.概述	05
2.产品优势	07
3.产品架构	08
4.性能指标	10
5.功能原理	11
5.1. 消息发送	11
5.2. 消息拉取	12
5.3. 事务消息	13
5.4. 顺序消息	14
5.5. 单元化功能	14
5.6. 消息路由	16
5.7. 用户管理	17
5.8. ACL 策略	17
5.9. 本地优先	17
6.基础术语	18

1.概述

SOFAShark Kafka 消息队列（SOFAShark Kafka，简称 SOFAKafka）是基于 Apache Kafka 构建的分布式消息中间件，并与金融分布式架构 SOFAShark 深度集成，为分布式应用系统提供异步解耦和削峰填谷的能力，支持顺序消息、事务消息等多种消息类型，并具备高可靠、高吞吐、低延时等金融级特性。

应用场景

数据监控

- 异步解耦消息队列的生产消费模型可以解耦上下游业务系统，并支持下游多个消费者对同一消息进行消费和处理。以金融场景为例，支付中心作为支付宝主站最核心的系统，每笔支付数据的产生会引起几百个下游业务系统的关注，包括账户中心、用户中心、权益中心、流计算分析等，整体业务系统庞大而且复杂，在应用强耦合的情况下，任一应用故障都将可能对业务带来影响。通过消息队列进行异步通信和应用解耦，可以很好的提升业务的连续性。
- 顺序收发指的是消息消费者按照消息发送的顺序进行消费，保证 FIFO。金融场景里需要保证顺序的应用场景非常多，例如证券交易过程中的时间优先原则，交易系统内的订单创建、支付、退款等流程等等。
- 分布式事务一致性应用解耦往往带来多个应用之间的事务一致性的问题。例如支付转账成功后，需要生成账单，更新用户积分等，此时通过消息队列的分布式事务处理功能，既可以实现系统之间的解耦，又可以保证最终的数据一致性。

日志处理

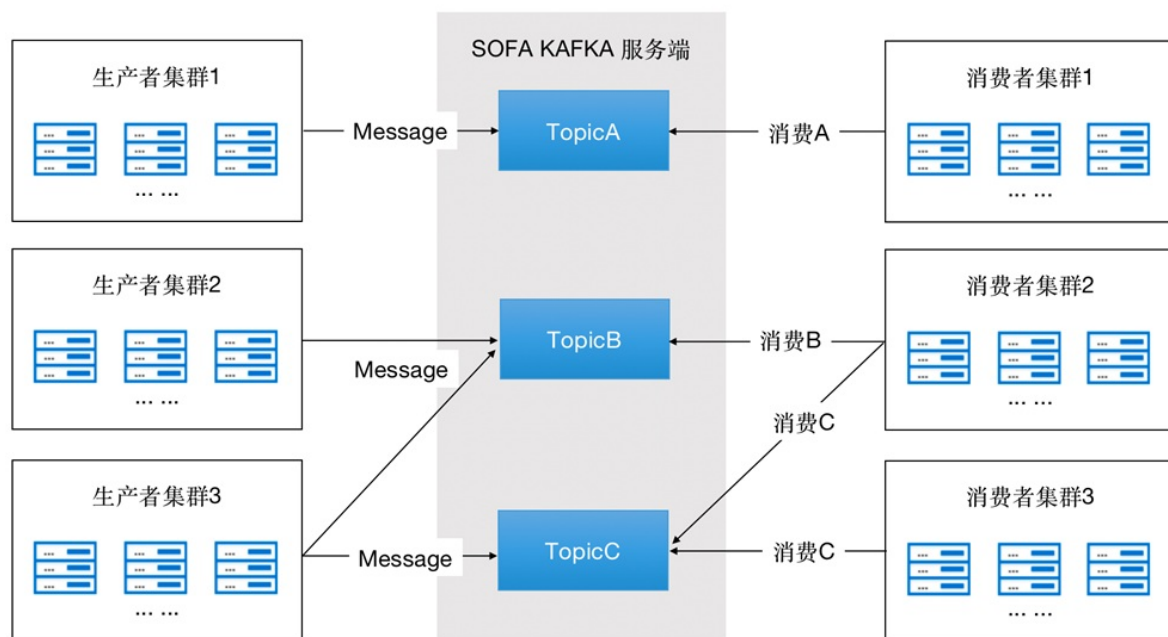
高吞吐使得它非常适合日志处理。应用可以批量异步地将日志发送到 SOFAKafka 集群里，而无需存在本地或 DB 里，消费者可以使用 spark 等分析系统对拉取到的日志做进一步统计分析。

大数据场景

- 削峰填谷应用分布式改造后，不同应用能承载的性能情况往往不一致，在诸如双 11、店庆、秒杀等大型活动时，将会带来较高的流量脉冲，部分系统可能导致系统超负荷甚至崩溃，影响用户体验，消息队列可提供强大的抗积压能力，实现削峰填谷，生产方生产消息后，消费方可以按照系统自身的承受能力进行消息的消费。
- SOFAKafka 的高吞吐、数据压缩、磁盘存储空间分配等方面的优势决定了它适合处理海量实时数据，并能汇总数据。具体场景有离线数据、流式数据的聚合分析等操作。

消息收发模型

消息队列支持生产/消费模型，消息生产者应用创建 Topic 并将消息发送到 Topic。消费者应用主动去拉取指定 Topic 的消息。通信可以是一对多（扇出）、多对一（扇入）和多对多。



生产者集群 用来表示发送消息应用，一个生产者集群下包含多个生产者实例，可以是多台机器，也可以是一台机器的多个进程，或者一个进程的多个生产者对象。一个生产者集群可以发送多个 Topic 消息。发送分布式事务消息时，如果生产者中途意外宕机，Broker 会主动回调生产者集群的任意一台机器来确认事务状态。

消费者集群 用来表示消费消息应用，一个消费者集群下包含多个消费者实例，可以是多台机器，也可以是多个进程，或者是一个进程的多个消费者对象。

一个消费者集群对应一个 Group ID，一个 Group ID 可以订阅多个 Topic，如上图中的消费者集群 2 所示。Group 和 Topic 的订阅关系可以直接在程序中设置。

一个消费者集群下的多个消费者以均摊方式消费消息。如果需要通过广播，只要每个消费者有一个独立的集群就可以了。如果需要实现单播，需要所有的消费者在同一个集群里。

2. 产品优势

SOFAKafka 的产品优势如下：

- 功能齐全
 - 支持多种消息类型：普通消息、分区顺序消息、事务消息
 - 支持多种消费模式：push/pull 模式、pub/sub 模式
 - 支持 Java SDK
- 运维体系便捷
 - 支持多维度消息查询
 - 支持全链路消息轨迹
- 性能优越
 - 海量消息堆积能力
 - 毫秒级端到端延迟
 - 千万级高并发处理能力
- 服务可靠
 - 服务高可用
 - 数据高可靠
 - 支持消息重投机制

3. 产品架构

产品架构

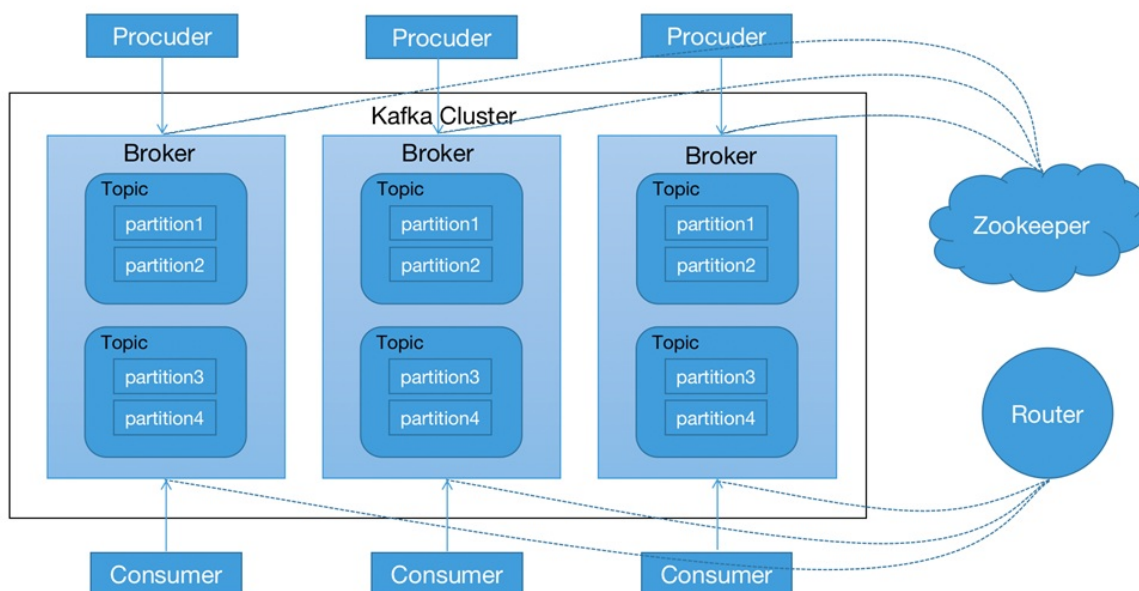
消息队列在任何一个环境都是可扩展的，生产者必须是一个集群，消息服务器必须是一个集群，消费者也同样。可选的 ACK 机制使得用户可以在性能和可靠性之间做取舍。

说明

白皮书中所提及的消息队列的服务端或者服务器包含 Zookeeper、Broker 等。服务端不等同于 Broker。

系统架构

系统部署架构如下图所示。



SOFAKafka 消息队列包括以下 6 个组件：

- **Zookeeper**：存储 SOFAKafka 集群的元数据信息，比如记录注册的 Broker 列表、Topic 元数据信息、partition 元数据信息等。
- **Broker**：消息中转角色，负责存储和转发消息。每个 Kafka 节点都是一个 Broker，Broker 存储了 Topic 的数据。一个 Topic 的数据可能存在多个 Broker 节点上，Topic 的数据会有多个副本，每个 Broker 节点既可能存储主副本数据也可能存储备份副本数据。
- **生产者 Producer**：消息发布者，与 Broker 集群建立长连接。主要作用是生产数据，并将产生的数据推送给 SOFAKafka 集群。
- **消费者 Consumer**：消息消费者，与 Broker 集群建立长连接。主要作用是消费 SOFAKafka 集群的数据。Consumer 既可以主动拉取 Broker 的数据也可以被动接收 Broker 推送数据。
- **Console**（未在图中展示）：SOFAKafka 的图形化管理控制台。
- **Router**：LDC 场景下，根据配置的 Router 规则，进行逻辑单元之间消息转发，使 SOFAKafka 支持 LDC 能力的组件。

网络架构

Broker 集群部署

- 为保证服务的可靠性，消息队列支持多节点集群化部署。
- 支持在多个机房部署。
- 支持集群化部署方式，提升部分节点不可用时的容灾能力。
- 消息重发机制。当某个节点服务不可用时，迅速切换到其他节点，提升集群的服务能力。

Broker 主备部署

- SOFAKafka 的每个 Broker 节点即可能是 Topic 分区的主副本也可能是从副本。
- SOFAKafka Broker 同时支持分区副本的主备间自动切换。一旦主副本所在的 Broker 出现不可用时，根据主备切换策略进行切换，迅速恢复服务。

4.性能指标

不同机器配置下的性能不同，即对于不同的 Pod 规格，系统吞吐量（TPS）会有所不同。详细的性能数据如下表所示：

Pod	存储磁盘	TPS（1KB 消息）
4C8G（3 台 Broker）	500GB SSD	5000
16C60G（3 台 Broker）	2TB SSD	20000

? 说明

一组 Broker 机器互为主备，如有需要，可以在此基础上以机器组的维度进行扩容。

5. 功能原理

5.1. 消息发送

TCP 协议支持三种消息发送方式：可靠同步发送、可靠异步发送、单向（Oneway）发送。本节介绍了每种发送方式的原理、适用场景以及三种发送方式的异同。

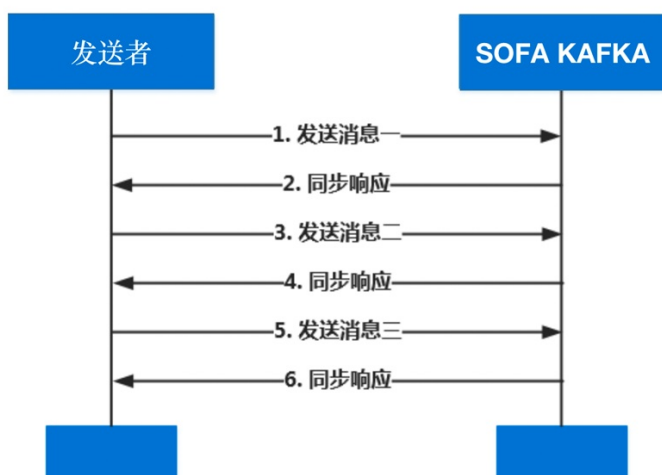
三种发送方式的对比

三者的特点和主要区别如下：

发送方式	发送 TPS	发送结果反馈	可靠性	适用场景
可靠同步发送	快	有	不丢失	此种方式应用场景非常广泛，例如重要通知邮件、报名短信通知、营销短信系统等。
可靠异步发送	快	有	不丢失	异步发送一般用于链路耗时较长，对响应时间较为敏感的业务场景，例如用户上传后通知启动转码服务，转码完成后通知推送转码结果等。
单向发送	最快	无	可能丢失	适用于某些耗时非常短，但对可靠性要求并不高的场景，例如日志收集。

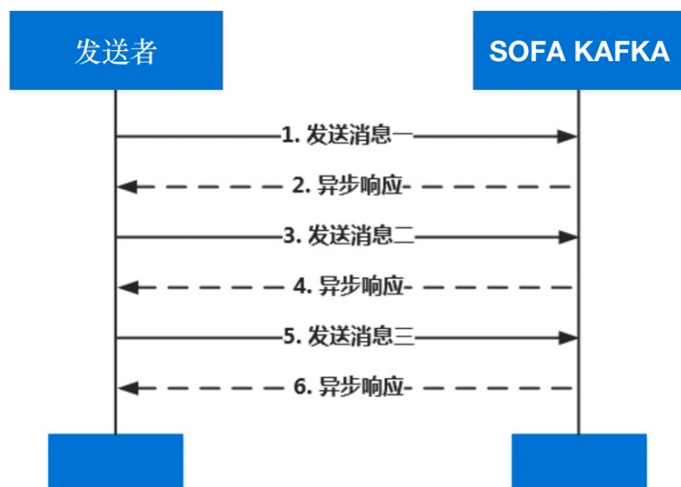
可靠同步发送

同步发送是指消息发送方发出数据后，会在收到接收方发回响应之后才发下一个数据包的通讯方式。



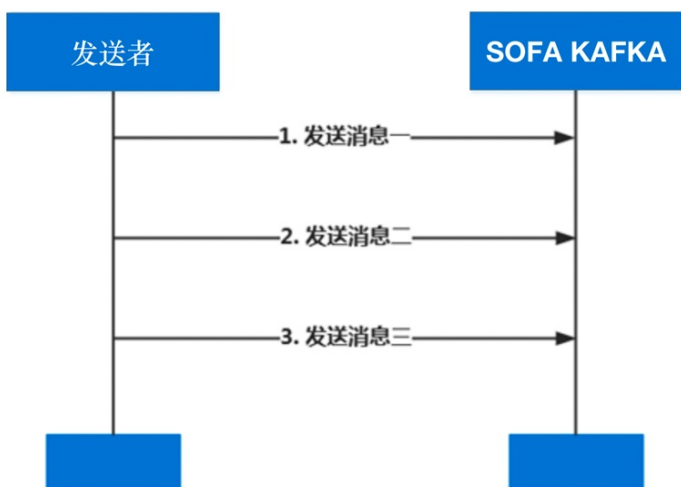
可靠异步发送

异步发送是指发送方发出一条消息后，不等待服务端返回响应，接着发送下一条消息的通讯方式。消息队列的异步发送，需要发送方实现异步发送回调接口（SendCallback）。消息发送方在发送了一条消息后，无需优化等待服务端响应即可发送第二条消息。发送方通过回调接口接收服务端响应，并处理响应结果。



单向（ONEWAY）发送

发送方只负责发送消息，不等待服务端返回响应且没有回调函数触发，即只发送请求不等待应答。此方式发送消息的过程耗时非常短，一般在微秒级别。

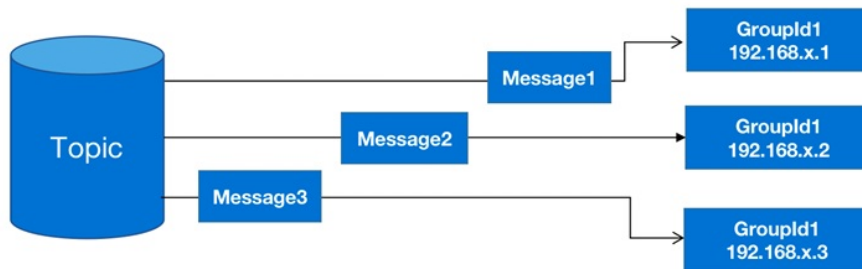


5.2. 消息拉取

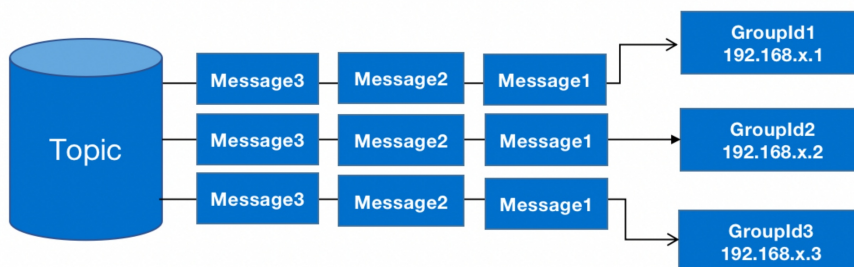
SOFAKafka 是基于生产消费模型的消息系统。在 SOFAKafka 消息系统中，消息的消费方通过拉取 Topic 以获取并消费消息。

由于消费应用一般是分布式系统，以集群方式部署有多台机器，因此 SOFAKafka 约定以下概念。

- **集群**：SOFAKafka 约定使用相同 GroupId 的消费者属于同一个集群，同一个集群下的消费者消费逻辑必须完全一致，这些消费者在逻辑上可以认为是消费节点。
- **集群消费**：当使用集群消费模式时，SOFAKafka 认为任意一条消息只需要被集群内的任意一个消费者处理即可。



- **广播消费**：当使广播消费模式时，集群内的所有客户端需要用不同的 GroupId 去消费数据，SOFAKafka 会将每条消息推送给集群内所有的客户端，保证消息至少被每台机器消费一次。



5.3. 事务消息

概念介绍

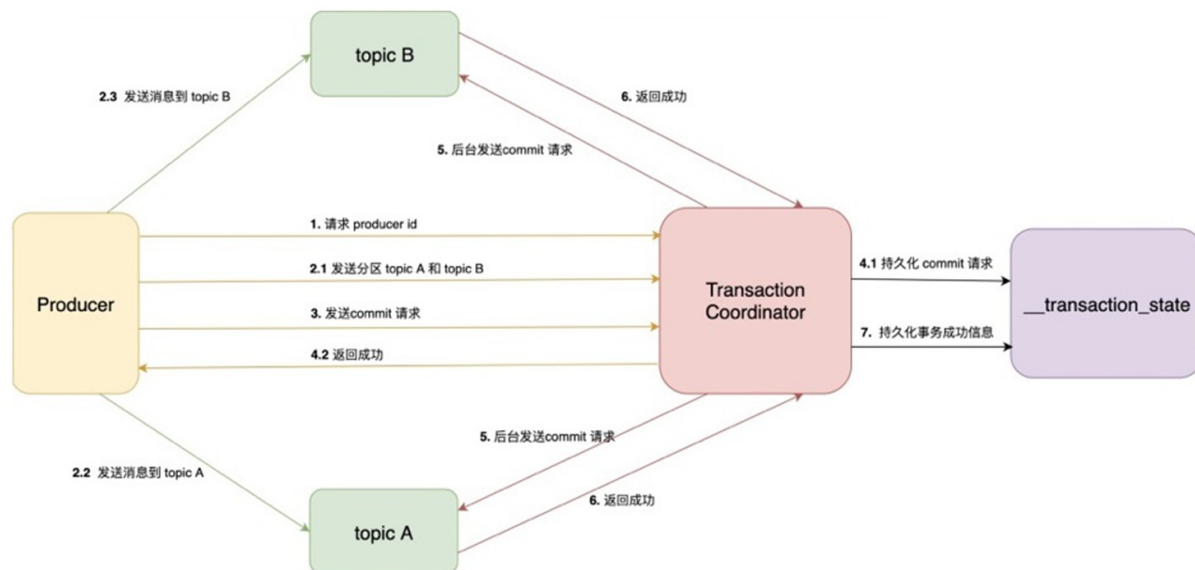
SOFAKafka 的事务不同于 SOFAMQ，SOFAMQ 是保障本地事务（比如数据库）与 SOFAMQ 消息发送的事务一致性，SOFAKafka 的事务主要是保障一次发送多条消息的事务一致性（要么同时成功要么同时失败）。

适用场景

事务消息的适用场景示例如下：

多用于流式计算场景，比如，SOFAKafka 需要对一个 Topic 里的消息做不同的流式计算处理，处理完分别发到不同的 Topic 里，这些 Topic 分别被不同的下游系统消费（比如 HBase、Redis、ES 等），此时就需要系统发送到多个 Topic 的数据保持事务一致性。

交互流程



事务消息发送步骤如下：

1. 创建 **Producer** 实例，并且指定 **Transaction ID**。
2. **initTransactions()**，初始化事务，向 **TC** 服务申请 **Producer ID**。
3. 发送消息，处理逻辑。
4. 发送提交请求，提交事务，**beginTransaction()**。
5. 提交请求持久化，提交偏移量，**sendOffsetsToTransaction()**。
6. 发送事务结果信息给分区，提交事务，**commitTransaction()**。

5.4. 顺序消息

顺序消息（FIFO 消息）是消息队列提供了一种严格按照顺序来发布和消费的消息。顺序发布和顺序消费是指对于指定的一个 Topic，生产者按照一定的先后顺序发布消息；消费者按照既定的先后顺序订阅消息，即先发布的消息一定会先被客户端接收到。

对于指定的一个 Topic，所有消息根据 **partition** 进行区块划分。同一个分区内的消息按照严格的 FIFO 顺序进行发布和消费。指定相同的 Key 也可以 hash 到同一个 **partition**。

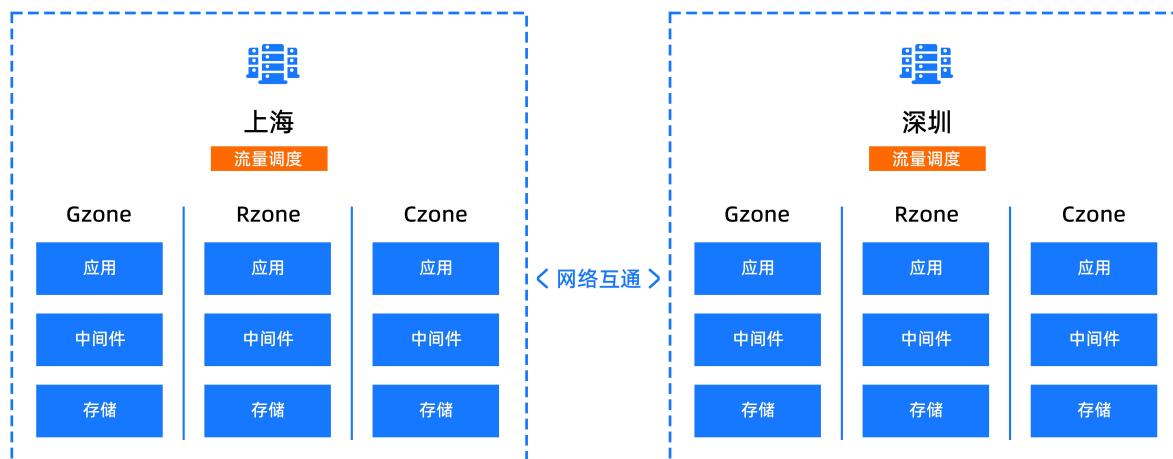
适用场景

适用于性能要求高，以 **Key** 或者 **partition** 作为分区字段，在同一个区块中严格地按照 FIFO 原则进行消息发布和消费的场景。

顺序消息的适用场景示例如下：用户注册需要发送验证码，以用户 ID 作为 **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。

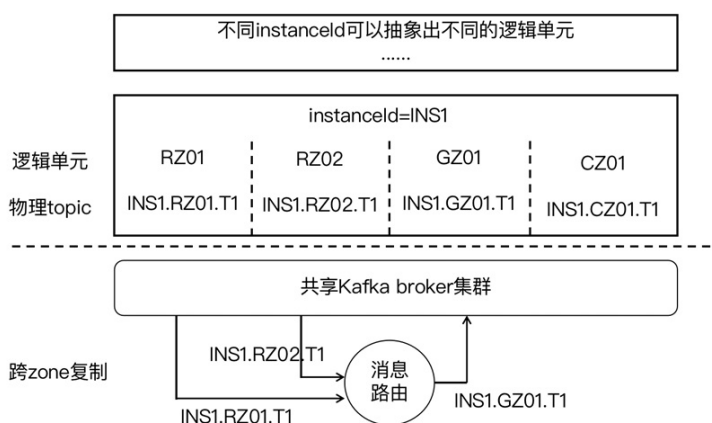
5.5. 单元化功能

SOFAKafka 的部署物理单元以城市维度部署。用户自行划分逻辑单元，共享使用 SOFAKafka 集群，优点是机器成本比较低，适合体量适中的用户。



实现原理

核心原理是通过消息路由（Router）组件实现消息复制。



Topic

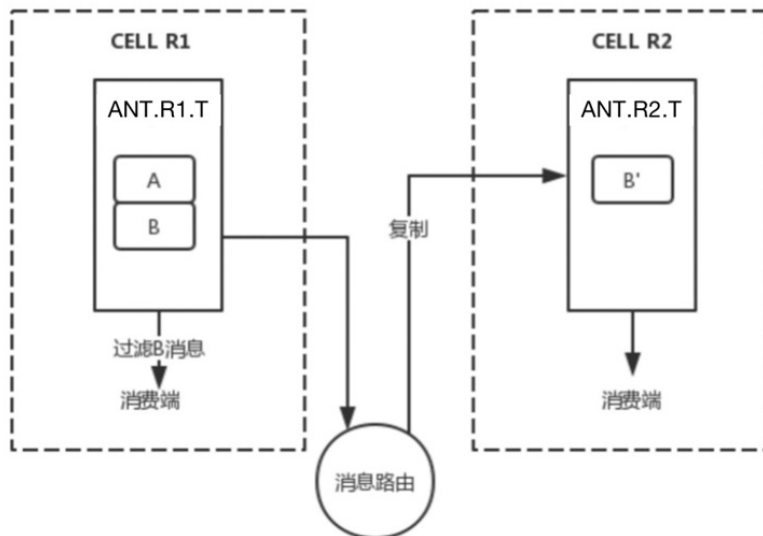
通过用户逻辑 Topic 和实际物理 Topic 的映射来实现逻辑单元消息隔离，每个逻辑 Zone 发送和消费都是各自的 Topic，数据是互相隔离的。

例如，InstanceId= ANT 应用在逻辑单元 CELL1 发送 Topic 为 T1 的消息，客户端实际发送的时候会将逻辑 Topic 转换成物理 Topic ANT.CELL1.T1。这样不同逻辑单元的应用，虽然在用户侧感知到的是同一个 Topic，实际上不同逻辑单元的物理 Topic 是不一样的。

Group

通过用户逻辑 Group 和实际物理 Group 的映射来实现逻辑单元消费的隔离，每个逻辑 Zone 发送和消费都是各自的 Group，事务回查、订阅关系和消费进度是互相隔离的。

例如，InstanceId= ANT 应用在逻辑单元 CELL1 消费通过 GroupG1 消费 Topic 为 T1 的消息，客户端实际消费的时候会将逻辑 Group 转换成物理 Group ANT.CELL1.G1，最终就是以 Group ANT.CELL1.G1 来消费 Topic ANT.CELL1.T1。



如上图示应用，对于在 R1 的应用，因为 B 不属于 R1，需要过滤掉 B 的账单消息。

对于在 R2 的应用，因为 B 的账单消息在 R1，如果要被 R2 的应用消费到的话，需要通过消息路由组件将的账单消息从 R1 复制到 R2。

从物理上来说，即将消息从 R1 对应的 Topic 复制到 R2 对应的 Topic。

LDC 实现原理

发送方实现

发送端发送消息时，客户端将逻辑 Topic 替换成物理 Topic `$NS|$CELL%$TOPIC`，发送消息会自动根据 `zonemng` 设置 `__UNIT.ORIUNIT`、`__UNIT.TARGET_G`、`__UNIT.TARGET_C`。如果设置了 `cellUid`，还会设置 `__UNIT.TARGET_R`。

Local 订阅

使用 `$NS.$CELL.$GROUP` 订阅 `$NS.$CELL.$TOPIC` 消费 `$NS.$CELL.$TOPIC` 里的数据，只消费 `__UNIT.ORIUNIT ==` 当前 CELL 的消息。

RZone 订阅

客户端只在 RZone 启动对应的订阅关系，使用 `$NS.$CELL.$GROUP` 订阅 `$NS.$CELL.$TOPIC`，只消费 `__UNIT.TARGET_R ==` 当前 CELL 的消息。

GZone 订阅

客户端只在 GZone 启动对应的订阅关系，使用 `$NS.$CELL.$GROUP` 订阅 `$NS.$CELL.$TOPIC`，只消费 `__UNIT.TARGET_G ==` 当前 CELL 的消息。

CZone 订阅

客户端只在 CZone 启动对应的订阅关系，使用 `$NS.$CELL.$GROUP` 订阅 `$NS.$CELL.$TOPIC`，只消费 `__UNIT.TARGET_C ==` 当前 CELL 的消息。

5.6. 消息路由

LDC 场景下，不同的 InstanceId 可以抽象成不同的逻辑单元。当应用被部署在服务器上后，您可以使用消息路由功能将 A 逻辑单元的消息转发到 B 逻辑单元。

适用场景

LDC 场景下，可以使用消息路由进行跨逻辑单元转发消息。

5.7. 用户管理

SOFAKafka 在不开启任何配置的情况下，默认没有加密或者身份验证，也没有配置 ACL，任何客户端都可以使用该端口与 SOFAKafka Broker 进行通信，此时的 SOFAKafka Broker 集群是完全开放状态，任何人都可以直接访问。SOFAKafka 支持通过 SASL 认证来完成用户认证功能，目前我们使用的是其中功能比较完善的 SASL/SCRAM 认证方式。

实现原理

SOFAKafka 底层通过命令创建 scram 用户，保存在元数据中，用户在添加 ACL 策略后，可以对某个资源收发消息。

适用场景

目前 SOFAKafka 所有的消息收发操作都需要创建用户，并给用户添加 ACL 策略，这样可以保证资源的安全性，即只有具备权限的用户才可以操作该资源，而没权限的用户则无法操作。

5.8. ACL 策略

ACL 即访问控制列表（Access Control List），当 SOFAKafka 不开启任何配置时，SOFAKafka Client 应用可以通过连接 Zookeeper 地址，来获取存储在 Zookeeper 内的元数据信息，从而可以连接到 Broker 操作所有元数据。SOFAKafka 通过 ACL 来完成权限控制功能，用户通过对用户与资源授权，来规范管理每个资源的使用权限。

实现原理

SOFAKafka 底层通过命令操作资源、用户和权限，创建一条 ACL 策略保存在元数据中，当用户对某个资源收发消息时，SOFAKafka 会查看当前用户是否具有操作当前资源的权限，只有具备操作权限时，才可以成功操作，否则会有报错提示。

适用场景

目前 SOFAKafka 所有的消息收发操作都会使用到 ACL 策略，保证资源的安全性，即只有具备权限的用户才可以操作该资源，而没权限的用户则无法操作该资源。

5.9. 本地优先

SOFAKafka 多机房部署时，开启本地优先后，Client 会优先往本机房的 Broker 发送消息，当本机房的 Broker 全部不可用后，才会发送到另一个机房。

适用场景

SOFAKafka 在多机房部署的场景下，建议开启本地优先模式，从而保证多机房各个集群直接可以连通，达到容灾的效果。

6.基础术语

本文主要对 SOFAKafka 消息队列涉及的专有名词及术语进行定义和解析，方便您更好地理解相关概念并使用消息队列。

中文	英文	释义
消息主题	Topic	消息主题，一级消息类型，通过 Topic 对消息进行分类。
主题分区	Partition	一个主题对应多个分区，生产者发送给 Topic 的消息会进入其中一个分区。
消息	Message	消息队列中信息传递的载体。
偏移量	offset	主题分区内的消息会分配从 0 开始的 offset 作为唯一标识。消费者根据 offset 标识自己当前消费位置。
Message Key	Key	消息的附带内容，相同的 Key 会 hash 到同一个 partition 内。
消息生产者	Producer	消息生产者，也称为消息发布者，负责生产并发送消息。
Producer 实例	Producer instance	Producer 的一个对象实例，不同的 Producer 实例可以运行在不同进程内或者不同机器上。Producer 实例线程安全，可在同一进程内多线程之间共享。
消息消费者	Consumer	消息消费者，也称为消息订阅者，负责接收并消费消息。
Consumer 实例	Consumer instance	Consumer 的一个对象实例，不同的 Consumer 实例可以运行在不同进程内或者不同机器上。一个 Consumer 实例内配置线程池消费消息。
Group	Group	一类 Consumer，这类 Consumer 通常消费同一类消息，且消费的逻辑一致。
Group ID	Group ID	Group 的标识。
队列	Queue	每个 Topic 下会由一到多个队列来存储消息。

中文	英文	释义
集群消费	Clustering consumption	一个 Group ID 所标识的所有 Consumer 平均分摊消费消息。例如某个 Topic 有 9 条消息，一个 Group ID 有 3 个 Consumer 实例，那么在集群消费模式下每个实例平均分摊，只消费其中的 3 条消息。
广播消费	Broadcasting consumption	一个集群的所有消费者都分配唯一的 Group ID。例如某个 Topic 有 9 条消息，一个集群有 3 个 Consumer 实例分配 3 个 Group ID，那么在广播消费模式下每个实例都会各自消费 9 条消息。
事务消息	Transactional message	消息队列提供类似 X/Open XA 的分布事务功能，通过消息队列的事务消息能达到分布式事务的最终一致。
顺序消息	Ordered message	消息队列提供的一种按照顺序进行发布和消费的消息类型，分为全局顺序消息和分区顺序消息，当前仅支持分区顺序消息。
分区顺序消息	Partitionally ordered message	对于指定的一个 Topic，所有消息根据 Sharding Key 进行区块分区。同一个分区内的消息按照严格的 FIFO 顺序进行发布和消费。Sharding Key 是顺序消息中用来区分不同分区的关键字段，和普通消息的 Message Key 是完全不同的概念。
消息堆积	Message accumulation	Producer 已经将消息发送到消息队列的服务端，但由于 Consumer 消费能力有限，未能在短时间内将所有消息正确消费掉，此时在消息队列的服务端保存着未被消费的消息，该状态即消息堆积。
消息过滤	Message filtering	Consumer 可以根据消息 Key 对消息进行过滤，确保 Consumer 最终只接收被过滤后的消息类型。消息过滤在消息队列的服务端完成。
重置消费位点	Reset consumption offset	以时间轴为坐标，在消息持久化存储的时间范围内（默认 3 天），重新设置 Consumer 对已订阅的 Topic 的消费进度，设置完成后 Consumer 将接收设定时间点之后由 Producer 发送到消息队列服务端的消息。