SOFAStack

消息队列 SOFAKafka 使用指南

产品版本：AntStack Plus 1.13.1

文档版本：20230707

蚂蚁集团 ANT GROUP

法律声明

蚂蚁集团版权所有©2022，并保留一切权利。

商标声明

免责声明

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 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.什么是 SOFAStack Kafka 消息队列? 06
1.1. 概述 06
1.2. 产品优势 06
1.3. 产品架构 07
1.4. 功能特性 07
1.5. 应用场景 09
1.6. 基础术语 09
2.快速入门 12
2.1. 创建 Topic 12
2.2. 发送消息 14
2.3. 查询消息 15
3.概览 20
3.1. 获取接入配置 20
4.Topic 管理 21
4.1. 创建 Topic 21
4.2. 查看 Topic 列表 23
4.3. 发送测试消息 23
4.4. 查看 Topic 分区状态 25
4.5. 查看 Topic 订阅关系 26
4.6. 编辑 Topic 28
4.7. 删除 Topic 28
5.Group 管理 29
5.1. 查看 Group 列表 29
5.2. 查看订阅关系 29
5.3. 查看消费者状态 31
5.4. 设置 offset 32

6.消息查询 35
7.消息路由 37
8.查看监控信息 40
9.ACL 策略管理 48
10.用户管理 50
11.Java SDK 参考 51
11.1. 准备环境 51
11.2. 参数说明 51
11.3. 收发送普通消息（两种方式） 54
11.4. 收发事务消息 58
11.5. 收发本地优先消息 62

1.什么是 SOFAStack Kafka 消息队列？

1.1. 概述

SOFAStack Kafka 消息队列（SOFAStack 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 SOFAStack 深度集成，广泛用于日志收集、监控数据聚合、流式数据处理、在线和离线分析等领域。

SOFAKafka 主要特点是基于 Pull 的模式来处理消息消费，追求高吞吐量。最初的目的就是用于日志收集和传输，适合产生大量数据的互联网服务的数据收集业务。而 SOFAMQ 具有高吞吐量、高可用性、适合大规模分布式系统应用的特点。相对 SOFAKafka，它对消息的可靠传输及事务性做了优化，目前被广泛应用于交易、充值、流计算、消息推送、日志流式处理、Binglog 分发等场景。

1.2. 产品优势

本文主要介绍 SOFAKafka 版相比于自建开源 ApacheKafka 所具备的优势。

开箱即用

- SOFAKafka 兼容社区版 Kafka 的 API，具备原生 Kafka 的所有消息处理特性。
- SOFAKafka 100% 兼容开源 Apache Kafka，您可以直接使用开源 Apache Kafka 客户端与 SOFAKafka 通讯。

> **重要**
>
> SOFAKafka 支持的客户端版本为 0.10.x~2.6.0，推荐客户端与服务端使用一致的版本。

高可用性

- 数据持久化：专业团队保障更高可用性，消息持久化落盘到消息队列，数据高可靠、服务高可用。
- 高吞吐能力：在海量消息堆积的情况下，始终能保持 SOFAKafka 集群的高吞吐能力。

数据安全

SOFAKafka 利用 SASL 机制对用户身份进行认证，并利用 SSL 对通道进行加密传输，确保数据在传输过程中不被窃取或篡改，保证您的数据安全。

高扩展性

您可以根据自身业务规模按需扩容，上层业务无感知。SOFAKafka 支持集群扩容和分区扩容两种模式。

- 集群扩容：支持横向扩容节点。
- 分区扩容：支持快速扩容分区。

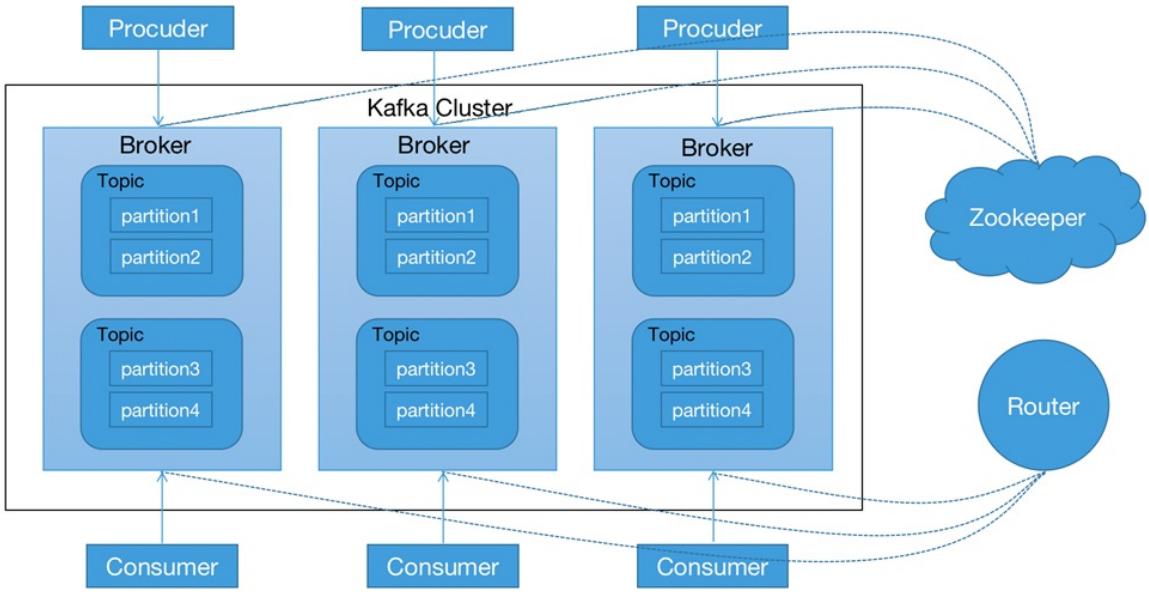
专业服务

SOFAKafka 拥有专业且经验丰富的运维团队，以及成熟的运维体系。监控报警：SOFAKafka 提供完整的监控图表和报警，帮助您及时发现问题。

1.3. 产品架构

本文介绍 SOFAKafka 的系统结构和发布/订阅模型。

系统部署架构如下图所示。



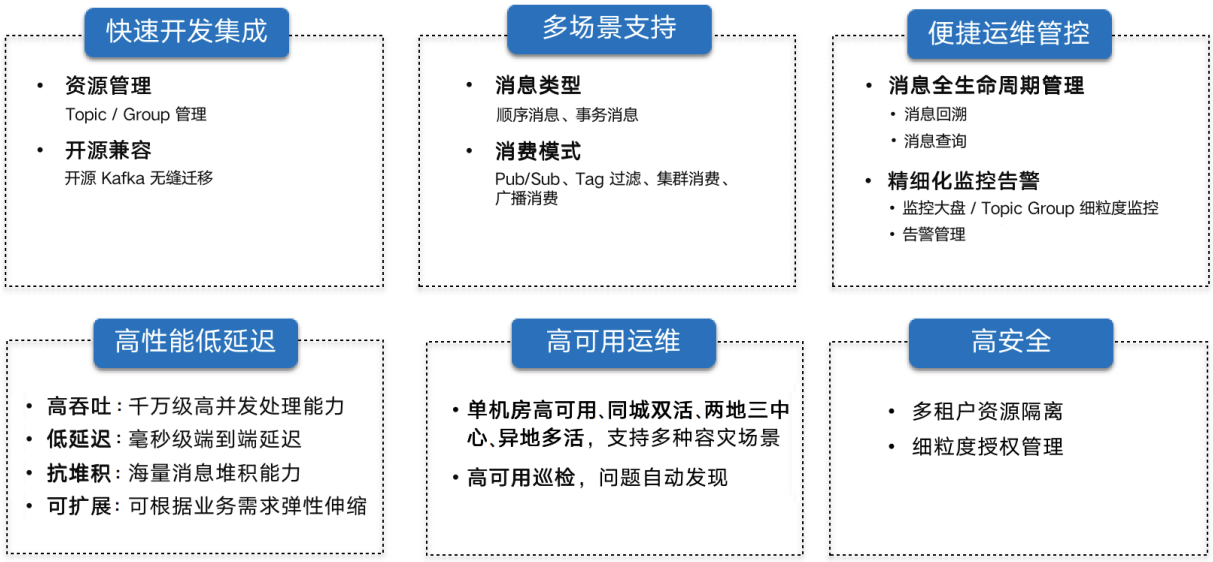
SOFAKafka 消息队列包括以下 6 个组件：

- **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。

发布/订阅模型

- **发布**：Consumer Group 和 Topic 的对应关系是 N : N，即一个 Consumer Group 可以同时订阅多个 Topic，一个 Topic 也可以被多个 Consumer Group 同时订阅。
- **订阅**：某个 Topic 的一条消息可以被多个 Consumer Group 同时订阅，但只能被同一个 Consumer Group 内的任意一个 Consumer 消费。

1.4. 功能特性



快速开发集成

资源管理

SOFAKafka 控制台支持 Topic 管理、Group 管理、消息查询，用户可以直接通过控制台发送消息。

开源兼容

100% 兼容开源 Apache Kafka，用户可以直接使用开源 Apache Kafka 客户端与 SOFAKafka 通讯。

多场景支持

消息类型

- 普通消息：消息队列中无特性的消息，区别于有特性的事务消息。
- 事务消息：实现类似 X/Open XA 的分布事务功能，以达到事务最终一致性状态。

消息模式

- 支持发布订阅模式。
- 支持集群消费和广播消费。
- 支持 Tag 过滤，消费者可以根据指定的 Tag 进行消息消费。

便捷运维管控

消息全生命周期管理

- 消息回溯：根据时间或位点重置消费进度，允许用户进行消息回溯或者丢弃堆积消息。
- 消息查询：支持按照位点或者时间查询消息。

精细化监控告警

提供完整的监控图表和报警，支持查看监控账户下创建的资源，包括实例、Topic、Consumer Group 等，帮助用户实时掌握资源状态，针对可能存在的问题及时处理，保障其稳定运行。

高性能低延迟

- 高吞吐：千万级高并发处理能力。
- 低延迟：毫秒级端到端延迟。

- 抗堆积：海量消息堆积能力。
- 可扩展：可根据业务需求弹性扩展。

高可用运维

- 支持多种容灾场景，支持单机房高可用、同城双活、两地三中心、异地多活部署架构。
- 高可用巡检，问题自动发现。

高安全

- 多租户资源隔离。
- 细粒度授权管理。

1.5. 应用场景

SOFAKafka 具有分布式、高吞吐、可扩展的特性，主要用于不同系统间的数据交流和传递，在企业解决方案、电商、物联网等众多领域都有广泛应用，其中主要包括以下应用场景。

日志收集

SOFAKafka 可以收集各种服务的 log，支持通过 SOFAKafka 以统一接口服务的方式开放给各种 Consumer。

消息系统

解耦生产者和消费者、缓存消息等。

用户活动跟踪

SOFAKafka 看用于记录 Web 用户或者 APP 用户的各种活动，如浏览网页、搜索、点击等活动，这些活动信息被各个服务器发布到 SOFAKafka 的 Topic 中，然后消费者通过订阅这些 Topic 来做实时的监控分析，也可以保存到数据库。

运营指标

SOFAKafka 可用于记录运营监控数据，包括收集各种分布式应用的数据，生产各种操作的集中反馈，比如报警和报告。

流式处理

支持 Spark Streaming 和 Storm 接入 SOFAKafka。在大数据相关的业务场景中，使得 SOFAKafka 能够支持流式数据的处理，并能方便的进行数据聚合。

1.6. 基础术语

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。

中文	英文	释义
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。
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。

中文	英文	释义
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。

2.快速入门

2.1. 创建 Topic

Topic 用于存储生产者发布的消息。在发送消息前，您必须先创建Topic。

1. 登录 SOFAKafka 控制台。
2. 在左侧导航栏上，单击 Topic 管理。
3. 单击 创建 Topic，然后在 创建 Topic 对话框配置 Topic 信息。

参数	是否必填	说明
名称	必填	Topic 名称，格式要求如下： ○ Topic 只能包含英文、数字、 短横线（-）和下划线（_），其中英文和数字必须要有一种， 短横线（-）和下划线（_）可选。 ○ 名称长度限制在3~128 字符之间。 ○ 命名不能以“CID”和“GID”开头。
分区数	必填	一个物理上分区的概念，一个 Topic 可以包含一个或者多个分区。分区数限制在 1~500。
副本数	必填	分区的副本个数，用于保障分区的高可用。 ○ 目前不支持创建单副本 Topic，默认开启 2 副本。 ○ 选择 n 个副本时，最多允许有（n-1）台 Broker 宕机。
描述	选填	对该 Topic 的备注信息，长度限制在 256 个字符以内。
高级配置		

参数	是否必填	说明
cleanup.policy	选填	清理日志策略。可选择 delete 或 compact。具体说明详见 cleanup.policy 参数说明。 ○ delete：日志按保存时间删除。 ○ compact：日志按 key 压缩。 说明 SOFAKafka connect 时需要使用 compact 模式。
min.insync.replicas	选填	当 producer 设置 ack（request.required.acks）为 -1 时，min.insync.replicas 指定了需要同步数据的最小副本数目。具体说明详见 min.insync.replicas 参数说明。
unclean.leader.election.enable	选填	指定是否能够设置不在 ISR 中 replicas 作为 leader。具体说明详见 unclean.leader.election.enable 参数说明。
segment.ms	选填	Segment 分片滚动的时长，单位为 ms，最小值为 86400000ms（1 天）。时长范围是 1天 ~ 90 天。具体说明详见 segment.ms 参数说明。
retention.ms	选填	Topic 维度的消息保留时间，单位为 ms，时间范围是 1 分钟 ~ 90 天。具体说明详见 retention.ms 参数说明。

4. 单击 **提交**。

2.2. 发送消息

在创建 Topic 后，您可以直接在 SOFAKafka 控制台向 Topic 发送消息。

操作步骤

1. 登录 SOFAKafka 控制台。
2. 在左侧导航栏上，单击 **Topic 管理**。
3. 单击目标 Topic **操作** 列下的 **发送测试消息**，在 **发送测试消息** 面板中配置以下信息：

参数	是否必填	说明
Key	选填	消息的业务标识，由消息生产者设置，唯一标识某个业务逻辑。
消息体	必填	需要发送的具体消息内容。

4. 单击 **确定**。

2.3. 查询消息

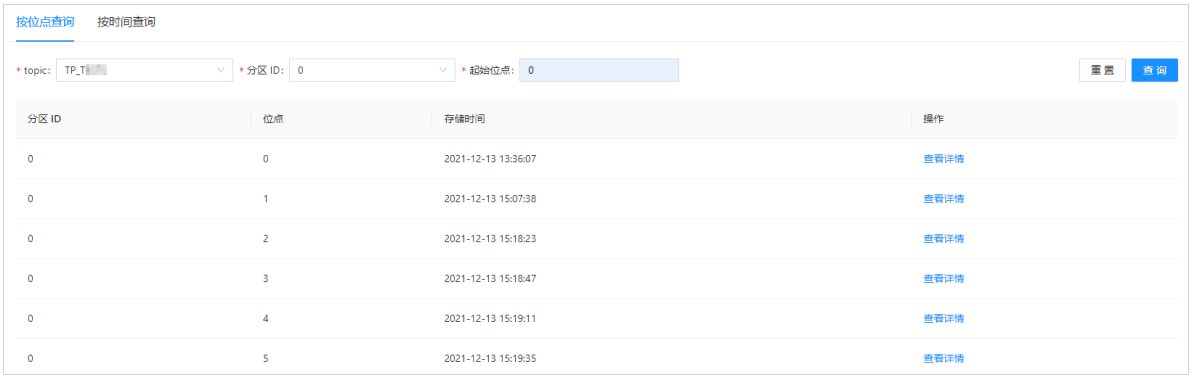
发送消息后，您可以查询发送的消息。

SOFAKafka 支持以下查询方式：

- **按位点查询**：按照位点来查询消息，适用于确定消息发送至 Topic 的分区 ID 以及消息位点。
- **按时间查询**：按照时间来查询消息，适用于不确定消息的位置，但是确定消息发送的时间。

按位点查询

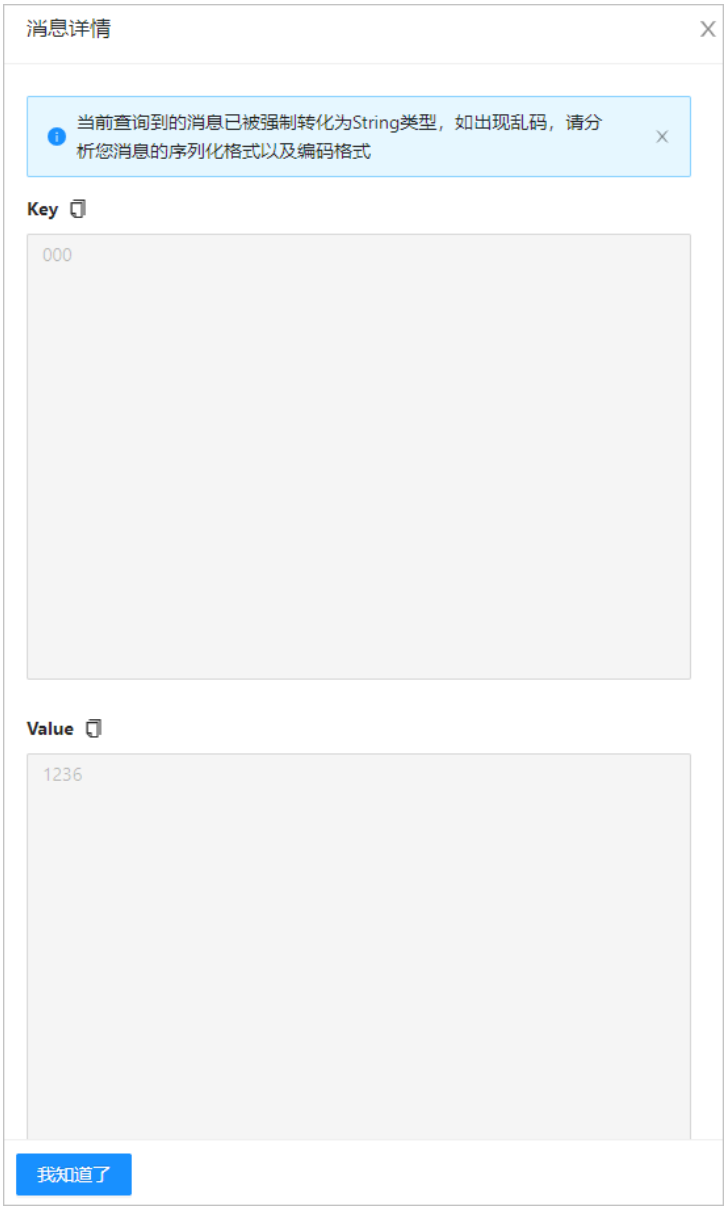
1. 登录 SOFAKafka 控制台。
2. 在左侧导航栏上，单击 **消息查询**。
3. 在 **按位点查询** 页签，选择 **Topic** 和 **分区 ID**，输入 **起始位点**，然后单击 **查询** 查看消息信息。



参数说明如下：

参数	说明
分区 ID	消息的 Topic 分区。
位点	消息的消费位点。
存储时间	服务端存储消息的时间。

4. 您可以单击目标位点 操作 列下的 查看详情 查看消息的 Key 和 Value。

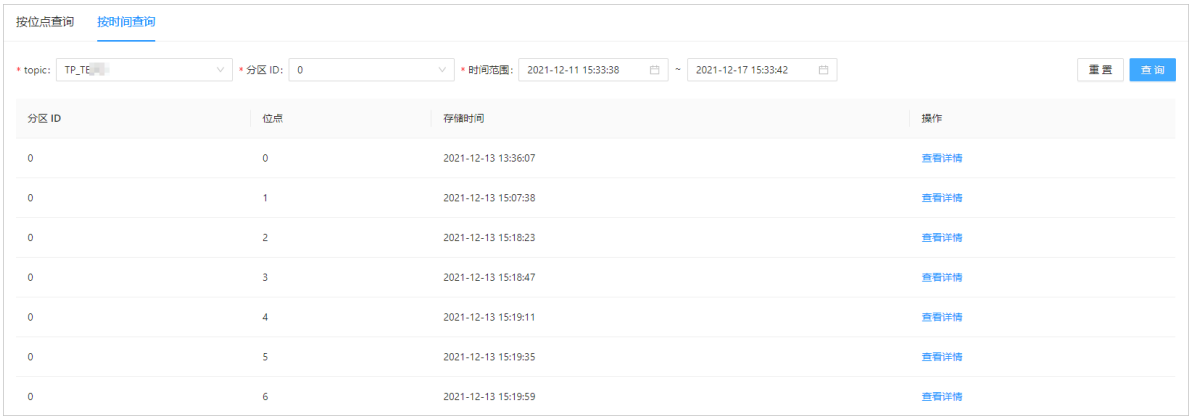


参数说明如下：

参数	说明
Key	消息的键（已强制转化为 String 类型）。
Value	消息的值（已强制转化为 String 类型）。

按时间查询

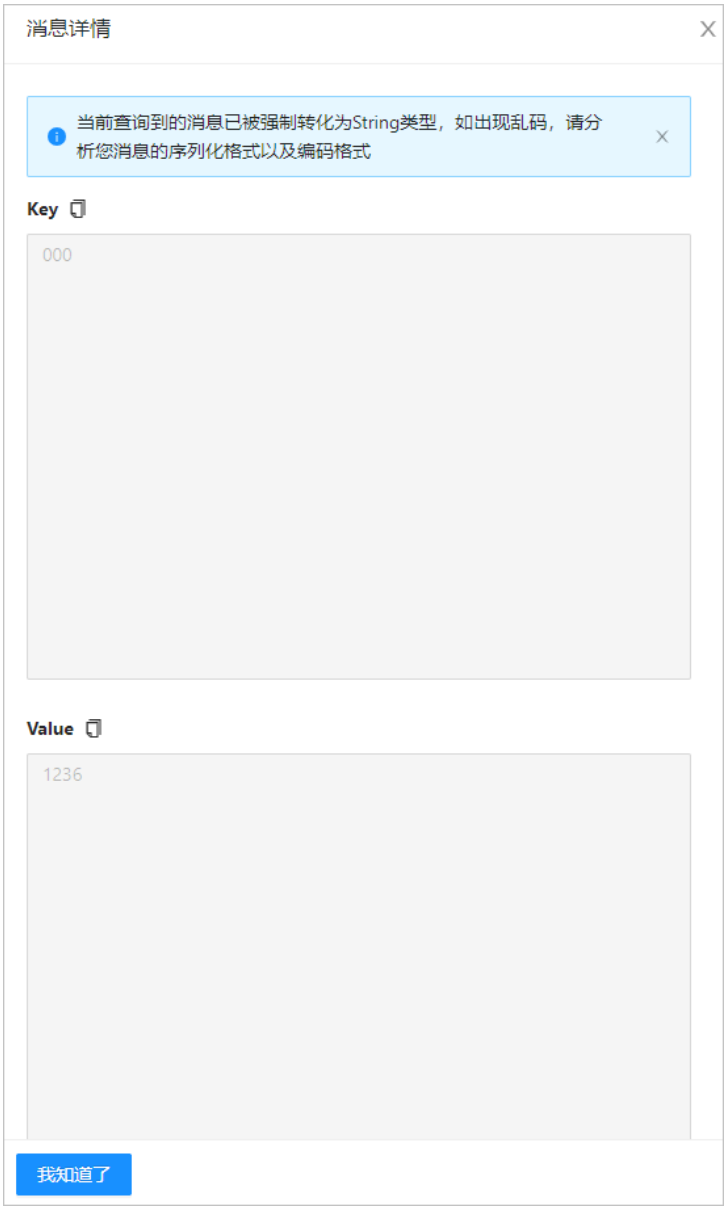
1. 在左侧导航栏上，单击 **消息查询**。
2. 在 **按时间查询** 页签，选择 **Topic** 、**分区 ID** 和 **时间范围**，然后单击 **查询** 查看消息信息。



参数说明如下：

参数	说明
分区 ID	消息的 Topic 分区。
位点	消息的消费位点。
存储时间	服务端存储消息的时间。

3. 您可以单击目标位点 **操作** 列下的 **查看详情** 查看消息的 Key 和 Value。



参数说明如下：

参数	说明
Key	消息的键（已强制转化为 String 类型）。
Value	消息的值（已强制转化为 String 类型）。

3.概览

3.1. 获取接入配置

1. 登录 SOFAKafka 控制台。
2. 在左侧导航栏上，单击 概览。
3. 在 接入配置 区域获取 实例 ID 和 内网接入点。

4. 将 实例 ID 配置到客户端 SDK 代码的 `INSTANCE_ID` 参数，将 内网接入点 配置到客户端 SDK 代码的 `ENDPOINT` 参数。

   客户端示例代码如下：

```
props.put(SofaKafkaProducerConfig.INSTANCE_ID, "000001");
props.put(SofaKafkaProducerConfig.ENDPOINT, "acvip://172.16.XX.XX/000001-SOFAKAFKA");
```

4.Topic 管理

4.1. 创建 Topic

Topic 用于存储生产者发布的消息。在发送消息前，您必须先创建Topic。

1. 登录 SOFAKafka 控制台。
2. 在左侧导航栏上，单击 Topic 管理。
3. 单击 创建 Topic，然后在 创建 Topic 对话框配置 Topic 信息。

参数	是否必填	说明
名称	必填	Topic 名称，格式要求如下： ○ Topic 只能包含英文、数字、 短横线（-）和下划线（_），其中英文和数字必须要有一种， 短横线（-）和下划线（_）可选。 ○ 名称长度限制在3~128 字符之间。 ○ 命名不能以“CID”和“GID”开头。
分区数	必填	一个物理上分区的概念，一个 Topic 可以包含一个或者多个分区。分区数限制在 1~500。
副本数	必填	分区的副本个数，用于保障分区的高可用。 ○ 目前不支持创建单副本 Topic，默认开启 2 副本。 ○ 选择 n 个副本时，最多允许有（n-1）台 Broker 宕机。
描述	选填	对该 Topic 的备注信息，长度限制在 256 个字符以内。
高级配置		
cleanup.policy	选填	清理日志策略。可选择 delete 或 compact。 ○ delete：日志按保存时间删除。 ○ compact：日志按 key 压缩。 说明 SOFAKafka connect 时需要使用 compact 模式。

参数	是否必填	说明
min.insync.replicas	选填	当 producer 设置 request.required.acks 为 -1 时，min.insync.replicas 指定 repilcas 的最小数目。
unclean.leader.election.enable	选填	指定是否能够设置不在 ISR 中 replicas 作为 leader。
segment.ms	选填	Segment 分片滚动的时长，单位为 ms，最小值为 86400000ms（1 天）。时长范围是 1天 ~ 90 天。
retention.ms	选填	Topic 维度的消息保留时间，单位为 ms，时间范围是 1 分钟 ~ 90 天。

4. 单击 **提交**。

4.2. 查看 Topic 列表

本节介绍如何查看 Topic 列表。

1. 登录 SOFAKafka 控制台。
2. 在左侧导航栏上，单击 **Topic 管理**。

   您可以在 Topic 管理页面查看 Topic 列表。

4.3. 发送测试消息

发送测试消息用于快速验证 Topic 资源的可用性，主要用作测试，发送成功之后可在消息查询查看发送情况。正式生产请使用 调用 SDK 发送消息。

操作步骤

1. 登录 SOFAKafka 控制台。
2. 在左侧导航栏上，单击 **Topic 管理**。
3. 单击目标 Topic 操作 列下的 **发送测试消息**。

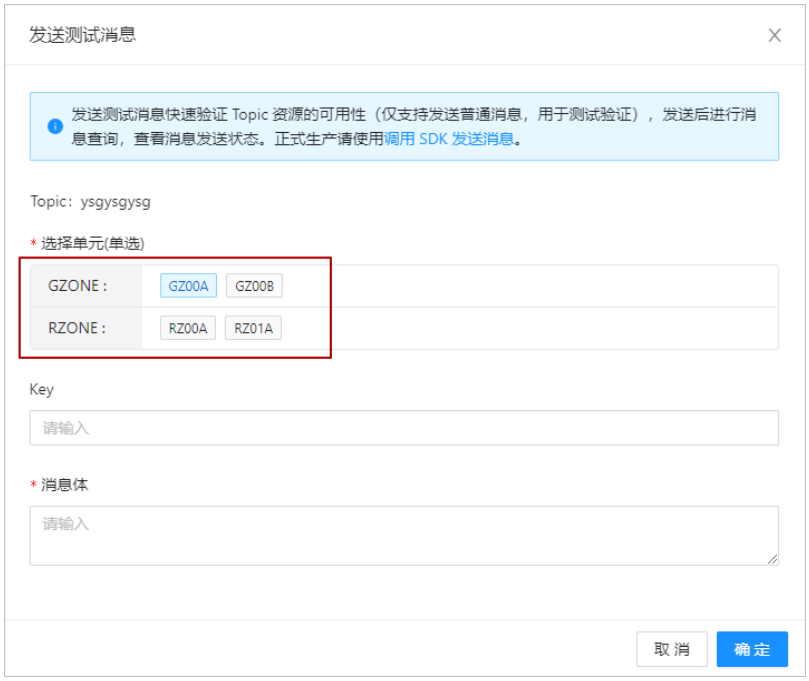
参数	是否必填	说明
Key	选填	消息的业务标识，由消息生产者设置，唯一标识某个业务逻辑。
消息体	必填	需要发送的具体消息内容。

4. 单击 **提交**，即可发送测试消息。

   消息发送后，您可以直接单击 **进入消息查询** 查看该消息的发送情况，或复制该消息的 MessageID 稍后手动查询该消息。有关消息查询的更多信息，参见 消息查询。

单元化说明

在 LDC 单元化架构环境下，可以自由切换单元发送测试消息，如下图所示。



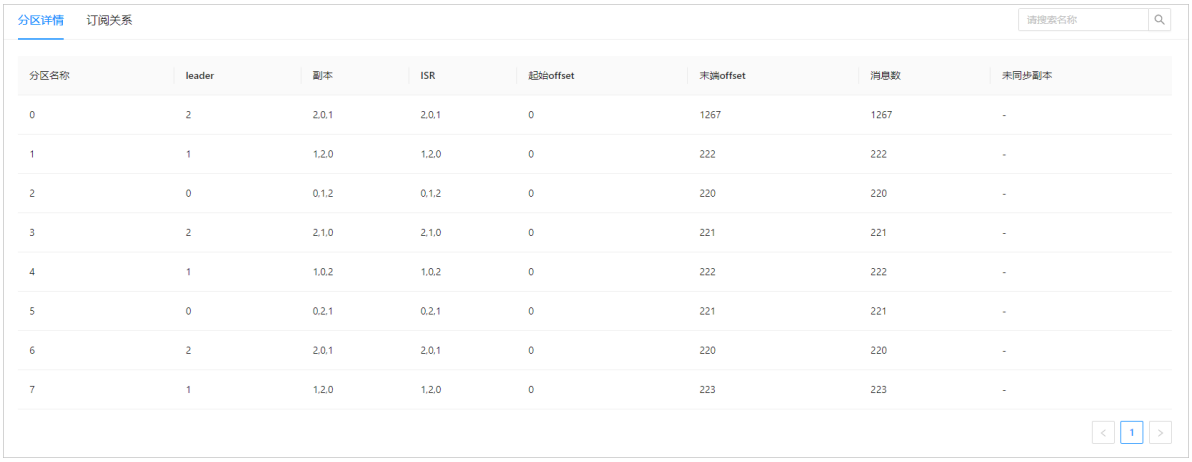
4.4. 查看 Topic 分区状态

创建 Topic 后，您可以在 **Topic 管理** 页面查看 Topic 的各个分区的状态。

操作步骤

1. 登录 SOFAKafka 控制台。
2. 在左侧导航栏上，单击 **Topic 管理**。
3. 单击目标 Topic 名称进入 Topic 详情页。

   您可以在 **分区详情** 页签查看 Topic 各个分区的详情。



参数说明如下：

参数	说明
分区名称	分区（partition）的名称。
leader	leader 处理 partition 的所有读写请求，follower 会被动定期地去复制 leader 上的数据。
副本	副本列表。
ISR	已同步消息的副本。
起始 offset	消息最后消费的位置。
末端 offset	消息最后写入的位置，若末端 offset 大于起始 offset，则代表有消息还没有被消费。
消息数	存储的消息数量。
未同步副本	未同步的副本数量。

单元化说明

在 LDC 单元化架构环境下，可以自由切换单元查看 Topic 分区状态，如下图所示。

4.5. 查看 Topic 订阅关系

创建 Topic 后，您可以在 **Topic 管理** 页面查看已订阅 Topic 的在线消费组。

操作步骤

1. 登录 SOFAKafka 控制台。
2. 在左侧导航栏上，单击 **Topic 管理**。
3. 单击目标 Topic 名称进入 Topic 详情页。

   您可以在 **订阅关系** 页签查看 Topic 各个分区的详情。

参数说明如下：

参数	说明
消费组名称	已订阅该 Topic 的消费组的名称。
状态	消费组的状态。 ○ **Stable**：消费组中各个消费者已经加入，处于稳定状态。 ○ **Empty**：消费组内当前没有任何成员。
均衡算法	消费者分区分配策略，默认为 **range**。 range 策略是针对 Topic 而言的，在进行分区分配时，为了尽可能保证所有消费者均匀的消费分区，会对同一个 Topic 中的 partition 按照序号排序，并对消费者按照字典顺序排序。然后用分区的个数除以消费者线程的总数来决定每个消费者线程消费几个分区。如果除不尽，那么前面几个消费者线程将会多消费一个分区。
未消费的消息条数	未被消费的消息数量。

单元化说明

在 LDC 单元化架构环境下，可以自由切换单元，查看指定单元下该 Topic 被哪些 Group ID 订阅了，如下图所示。

4.6. 编辑 Topic

创建 Topic 后，您可以在 **Topic 管理** 页面修改 Topic 的配置。

> **说明**
>
> 除 **名称**、**分区数**、**副本数** 不支持修改，其余信息均可修改。

1. 登录 SOFAKafka 控制台。
2. 在左侧导航栏上，单击 **Topic 管理**。
3. 单击目标 Topic 操作 列下的 **编辑** 修改 Topic 信息。

   参数说明详情 创建 Topic。
4. 单击 **提交**。

4.7. 删除 Topic

创建 Topic 后，您可以在 **Topic 管理** 页面删除 Topic。

> **重要**
>
> 删除 Topic 时，存储在该 Topic 中的消息也将被删除，请谨慎操作。

1. 登录 SOFAKafka 控制台。
2. 在左侧导航栏上，单击 **Topic 管理**。
3. 单击目标 Topic 操作 列下的 **删除**，并在弹出的对话框中单击 **确定**。

5.Group 管理

5.1. 查看 Group 列表

本节介绍如何查看 Group 列表。

操作步骤

1. 登录 SOFAKafka 控制台。
2. 在左侧导航栏上，单击 **Group 管理**。

   您可以在 **Group管理** 页面查看 Group 列表。

单元化说明

在单元化架构环境下，可以自由切换单元，查看指定单元下的 Group 列表。

5.2. 查看订阅关系

操作步骤

1. 登录 SOFAKafka 控制台。
2. 在左侧导航栏上，单击 **Group 管理**。
3. 单击目标消费组名称进入 **Group 详情** 页。

   您可以在 **订阅关系** 页签查看该消费组订阅的 Topic 的信息。

参数说明如下：

参数	说明
分区名称	Topic 的分区序号。
提交的 offset 位置	消息消费位点，即该 Topic 在当前分区下的消息消费位点。
最大的 offset 位置	该 Topic 在当前分区下的最大消息消费位点。
未消费的消息条数	当前分区下的消息堆积总量（最大的 offset 位置减去提交的 offset 位置）。

单元化说明

在 LDC 单元化架构环境下，可以自由切换单元，查看指定单元下该消费组订阅的 Topic，如下图所示。

5.3. 查看消费者状态

操作步骤

1. 登录 SOFAKafka 控制台。
2. 在左侧导航栏上，单击 **Group 管理**。
3. 单击目标消费组名称进入 **Group 详情** 页。

   您可以在 **消费者状态** 页签查看该消费组下的消费者状态。

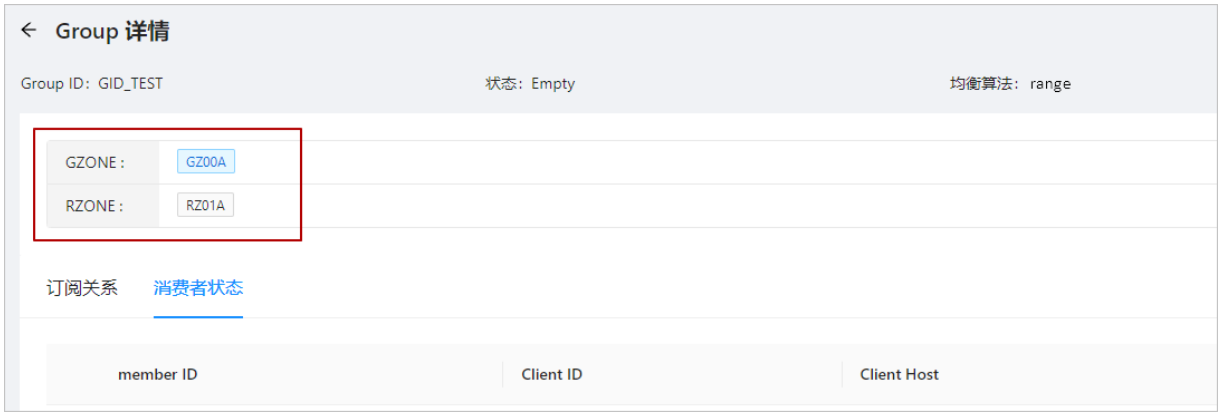
参数说明如下：

参数	说明
member ID	Coordinator 为消费分组中的消费者生成的唯一 ID。
Client ID	客户消费者 SDK 自己设置的 `client.id` 信息。
Client Host	消费者客户端的 IP 地址。
Topic 名称	Topic 的名称。
分区	Topic 的分区汇总。
分区名称	Topic 的分区序号。

参数	说明
提交的 offset 位置	消息消费位点，即该 Topic 在当前分区下的消息消费位点。
最大的 offset 位置	该 Topic 在当前分区下的最大消息消费位点。
未消费的消息条数	当前分区下的消息堆积总量（最大的 offset 位置减去提交的 offset 位置）。

单元化说明

在 LDC 单元化架构环境下，可以自由切换单元，查看消费者状态，如下图所示。



5.4. 设置 offset

设置 offset 是指改变订阅者当前的消费位置。您可通过设置 offset，按需清除堆积的或不想消费的这部分消息再开始消费，或直接跳转到某个时间点消费该时间点之后的消息（不论是否消费过该时间点之前的消息）。

1. 登录 SOFAKafka 控制台。
2. 在左侧导航栏上，单击 **Group 管理**。
3. 单击目标消费组名称 **操作** 列下的 **offset 设置**，在 **offset 设置** 对话框配置以下信息：

i. 选择需要重置 offset 的 Topic 和分区。

ii. 设置 offset。

SOFAKafka 支持以下 4 种重置 offset 方式：

- **将 offset 移动到指定位置**：输入需要重置至的位点，offset 设置范围要在最小 offset 和最大 offset 之间。在配置时，如果小于最小 offset 会从最小 offset 进行消费，如果大于最大 offset 会从最大 offset 进行消费。
- **将 offset 向前或向后移动若干条**：输入需要移动的条数，负数向前移动，正数向后移动。若移动后的位点小于当前分区的最小有效位点，则默认重置到最小有效位点；若移动后大于当前分区的最大有效位点，则默认重置到最大有效位点。
- **从最小/最开始位置开始消费**：选择从最新位置或最开始位置开始消费。
- **按时间点进行消费位置重置**：选择时间点，重置后消费组将从该时间点之后的消息开始消费。

4. 单击 **提交**。

6.消息查询

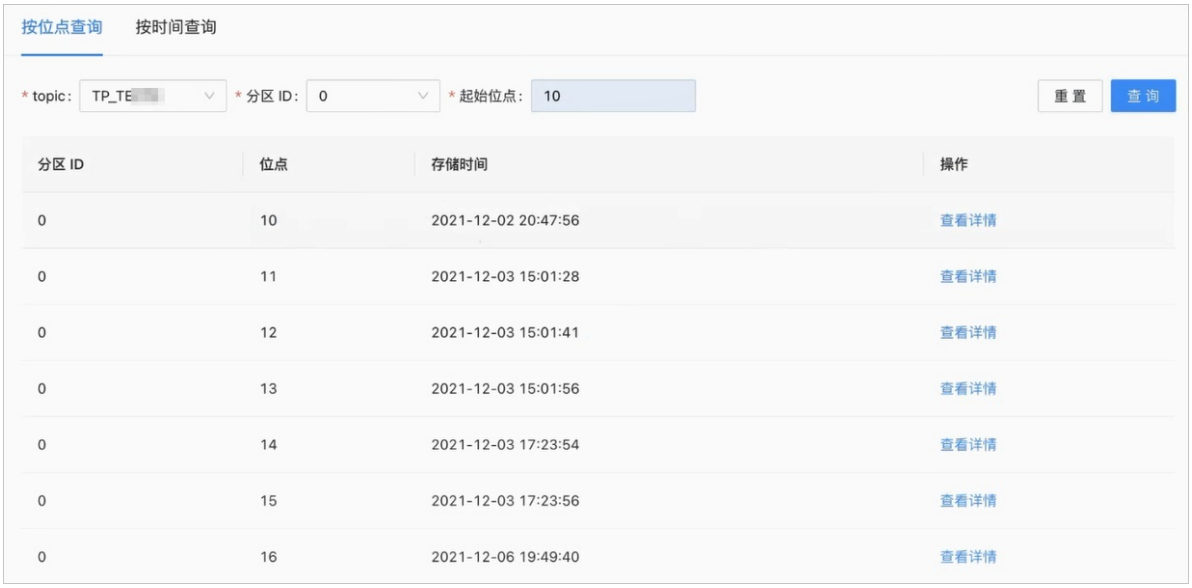
若您遇到消息消费异常，可以在 SOFAKafka 控制台查询异常消息来排查问题。

SOFAKafka 支持以下两种查询方式：

- **按位点查询**：按照位点来查询消息，适用于确定发送的消息的分区和位点。
- **按时间查询**：按照时间来查询消息，适用于不确定消息的位置，但是确定消息发送的时间。

按位点查询

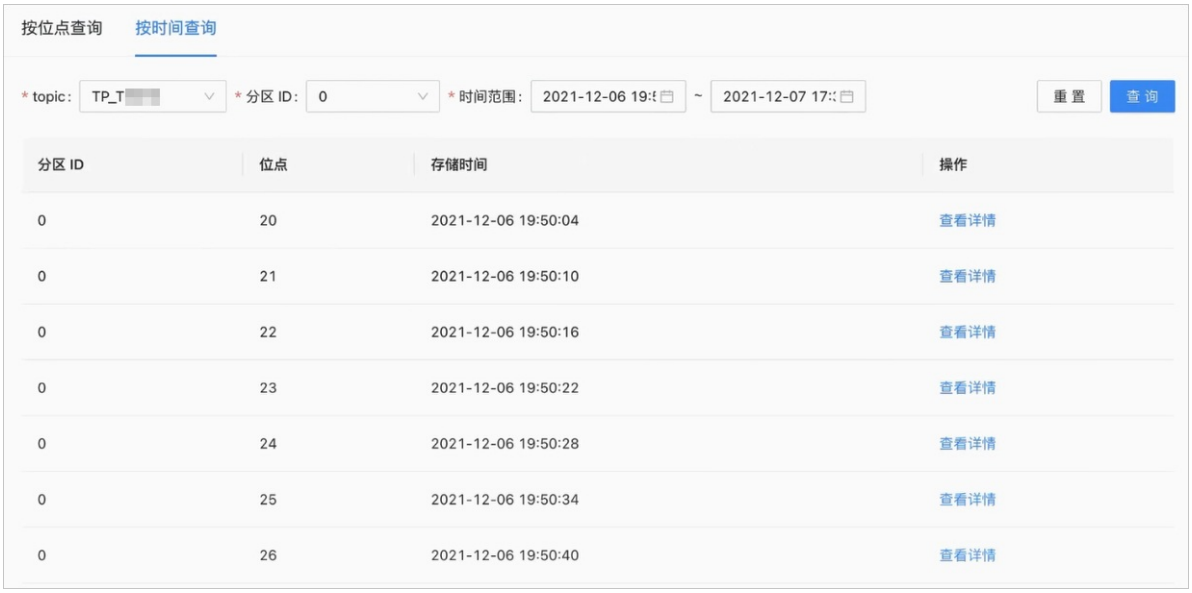
1. 登录 SOFAKafka 控制台。
2. 在左侧导航栏上，单击 **消息查询**。
3. 在 **消息查询** 页面，选择 **按位点查询** 页签，选择需要查询的 **topic**、**分区 ID** 和 **起始位点**。
4. 单击 **查询** 查看消息信息。



单击 **查看详情** 即可查看消息的 **Key** 和 **Value**。

按时间查询

1. 在左侧导航栏上，单击 **消息查询**。
2. 在 **消息查询** 页面，选择 **按时间查询** 页签，选择需要查询的 **topic**、**分区 ID** 和 **时间范围**。
3. 单击 **查询** 查看消息信息。



单击 **查看详情** 即可查看消息的 **Key** 和 **Value**。

单元化说明

在 LDC 单元化架构环境下，可以自由切换单元，在指定单元内进行消息查询，如下图所示。

7.消息路由

本文将引导您快速创建并管理消息路由任务，实现单元化部署环境下跨单元消息路由和同步。

> **说明**
>
> 该功能仅适用于支持 LDC 单元化架构的环境。

创建路由任务

您可以登录 SOFAKafka 控制台，根据业务需求创建跨地域的消息同步任务。操作步骤如下：

1. 登录 SOFAKafka 控制台。
2. 在左侧导航栏上，单击 **消息路由**。
3. 单击 **创建路由任务**，然后在 **创建路由任务** 对话框配置路由任务信息。

参数	说明
源 Topic	输入需要同步的消息所属 Topic 名称。
目标单元	选择消息将被同步到的 Topic 所属单元。

参数	说明
目标 Topic	输入消息将被同步到的 Topic 名称。
起始同步位点	选择从源 Topic 中的消息队列的哪个位置开始进行消息同步，即从这个位置之后进入队列的消息都会被同步到目标 Topic。支持选择 **最开始位点** 和 **最新位点**。 ○ **最开始位点**：任务首次启动之后，从有效期内最早写入源 Topic 队列的消息开始同步，首次任务启动之前发的消息不会被同步。 ○ **最新位点**：任务首次启动之后，从最新写入源 Topic 队列的消息开始同步，首次任务启动之前发的消息不会被同步。
描述	选填，输入对该同步任务的具体描述或备注。

4. 单击 **确定**。

任务创建完成后，即可在 **消息路由** 页面的任务列表中看到刚才创建的任务。

> **说明**
>
> ○ 消息路由任务创建成功后，该任务将处于 **初始化** 状态，需要等待片刻，完成初始化后将会进入 **运行中** 状态。
>
> ○ 配置路由转发的消息在生产者的 Zone 是默认能消费到的。配置了 GZone 路由转发的消息可以在 GZone 被消费到，配置了 RZone 路由转发的消息可以在 GZone 和指定的 RZone 被消费到。

启动、停止路由任务

在 **消息路由** 页面，单击目标任务操作列的 **启动** 或 **停止**，即可启动或停止该任务。

- 任务启动后，该任务的状态会切换至运行中。
- 任务停止后，该任务的状态会切换至已停止。

删除路由任务

> **说明**
>
> 仅支持删除已停止的消息路由任务。

在 **消息路由** 页面，单击目标任务操作列的 **删除**，即可删除该任务。

8.查看监控信息

SOFAKafka 支持查看监控账户下创建的资源，包括实例、Topic、Consumer Group 等，帮助您实时掌握资源状态，针对可能存在的问题及时处理，保障其稳定运行。SOFAKafka 还支持高级监控功能，您可以在控制台查看核心服务、实例资源和 Broker GC 等指标，方便运维人员在使用时进行排障处理。本节介绍如何查看监控指标，并介绍监控指标的含义。

操作步骤

1. 登录 SOFAKafka 控制台。
2. 在左侧导航栏上，单击 **监控管理**。

   您可以在 **监控管理** 页面查看实例、Topic、Consumer Group、高级监控的监控信息详细图表。

> **说明**
>
> - 采集周期包括 1 小时、3 小时、12 小时、1 天。采集周期决定了数据采集的时间间隔，周期越短，采集点越密集，生产消息量越详细。
> - 查询结果以图表形式显示。

实例

参数说明如下：

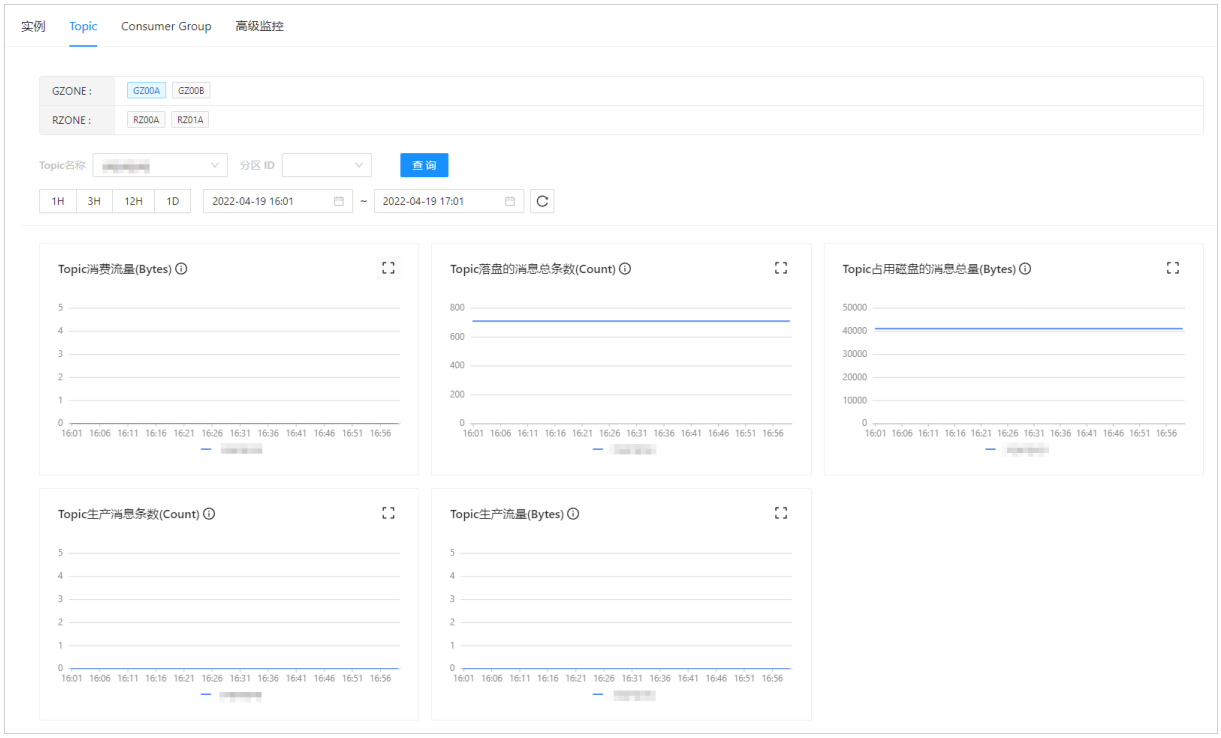
参数	说明
实例磁盘占用量（Bytes）	实例磁盘占用量（包含副本），按照所选择的时间粒度取最新值。

参数	说明
实例落盘的消息总条数（Count）	实例落盘的消息总条数（不包含副本），按照所选择的时间粒度取最新值。
实例消费流量（Bytes）	实例消费流量（不包含副本产生的流量），按照所选择的时间粒度统计求和。
实例生产消息条数（Count）	实例生产消息条数，按照所选择的时间粒度统计求和。
实例生产流量（Bytes）	实例生产流量（不包含副本产生的流量），按照所选择的时间粒度统计求和。

Topic

查询 Topic 的监控数据

不选择 分区 ID 时默认展示现有的 Topic 级别的监控数据。



参数说明如下：

参数	说明

参数	说明
Topic 消费流量（Bytes）	Topic 的消费流量（不包含副本产生的流量），按照所选择的时间粒度统计求和。
Topic 落盘的消息总条数（Count）	Topic 落盘的消息总条数（不包含副本），按照所选择的时间粒度取最新值。
Topic 占用磁盘的消息总量（Bytes）	Topic 占用磁盘的消息总量（不包含副本），按照所选择的时间粒度取最新值。
Topic 生产消息总数（Count）	Topic 生产消息条数，按照所选择的时间粒度统计求和。
Topic 生产流量（Bytes）	Topic 生产流量（不包含副本产生的流量），按照所选择的时间粒度统计求和。

查询 Topic 分区的监控数据

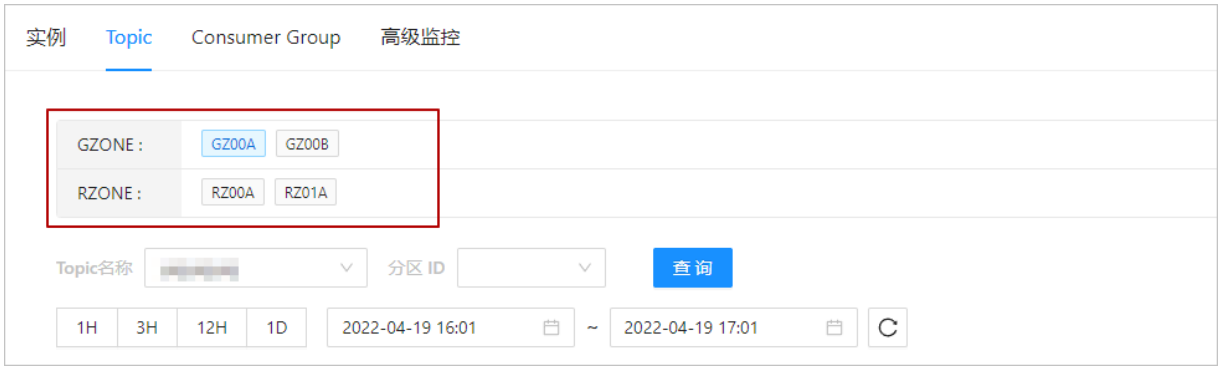
选择 **分区 ID** 后，您可以查看指定 Partition 的监控数据。

参数说明如下：

参数	说明
Partition 落盘的消息总条数（Count）	Topic 具体 Partition 的落盘的消息总条数（不包含副本），按照所选择的时间粒度取最新值。
Partition 占用磁盘的消息总量（Bytes）	Topic 具体 Partition 实际占用磁盘的消息总量（不包含副本），按照所选择的时间粒度取最新值。

查询指定单元的监控数据

在 LDC 单元化架构环境下，可以自由切换单元，查询 Topic 及其分区的监控数据，如下图所示。



Consumer Group

查询 Consumer Group 的监控数据

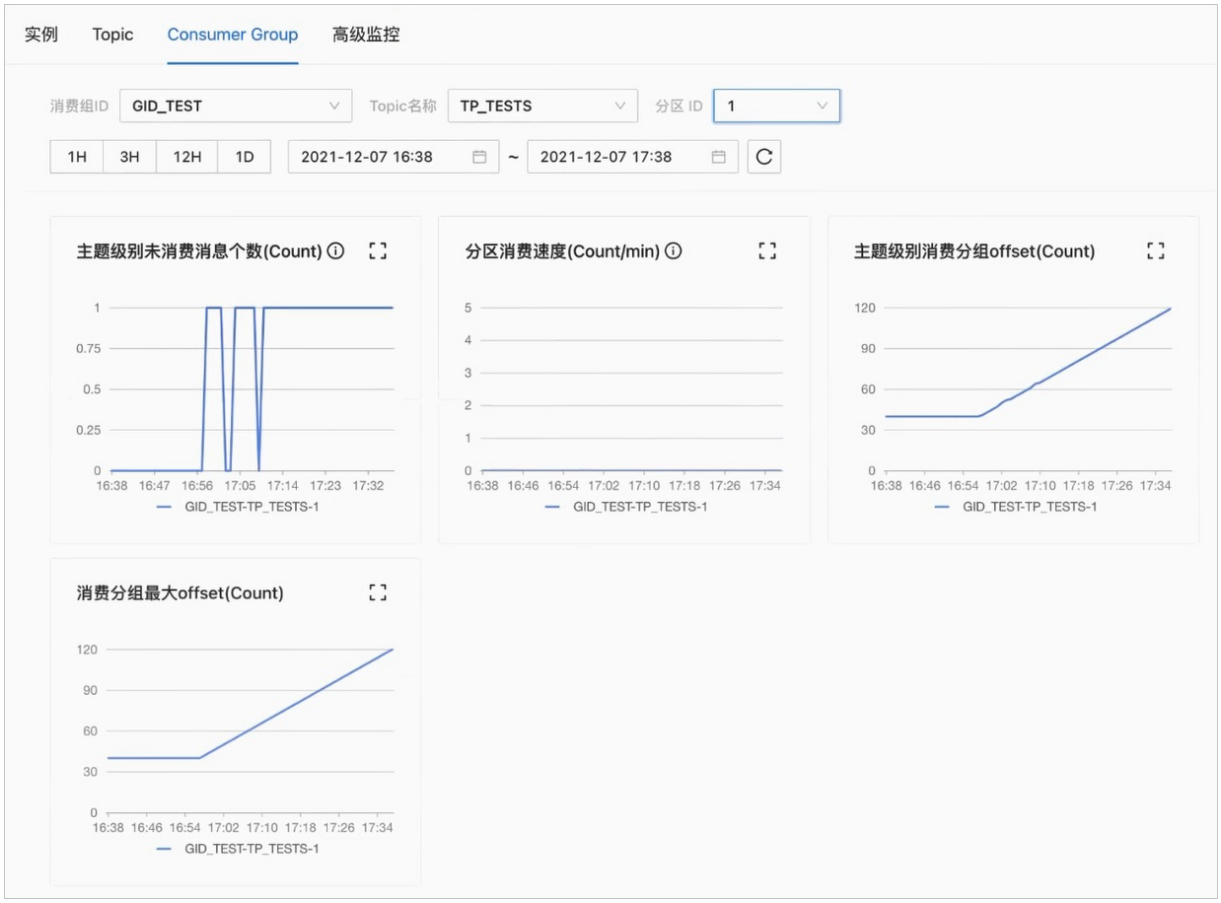
不选择 分区 ID 时默认展示现有的 Topic 级别的监控数据。

参数说明如下：

参数	说明
消费分组未消费消息数（Count）	Consumer Group 未消费消息条数，按照所选择的时间粒度统计求和。
主题消费速度（Count/min）	Consumer Group 主题消费速度。

查询 Consumer Group 分区的监控数据

选择 **分区 ID** 后，您可以查看指定 Partition 的监控数据。



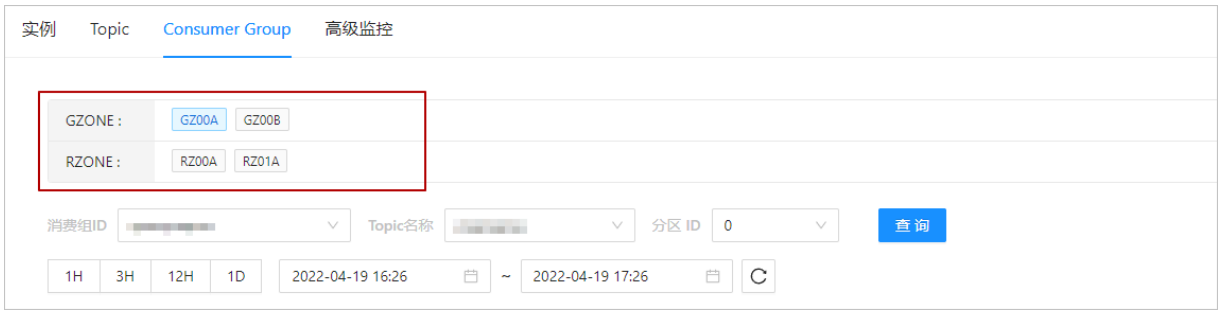
参数说明如下：

参数	说明
主题级别未消费消息个数（Count）	Consumer Group 在该分区下未消费消息数，按照所选择的时间粒度统计求和。
分区消费速度（Count/min）	消费分组在该分区的消费速率（条/分钟）。

参数	说明
主题级别消费分组 offset（Count）	消费分组该分区当前消费 offset，按照所选择的时间粒度统计求和。
消费分组最大 offset（Count）	当前分区最大 offset，按照所选择的时间粒度统计求和。

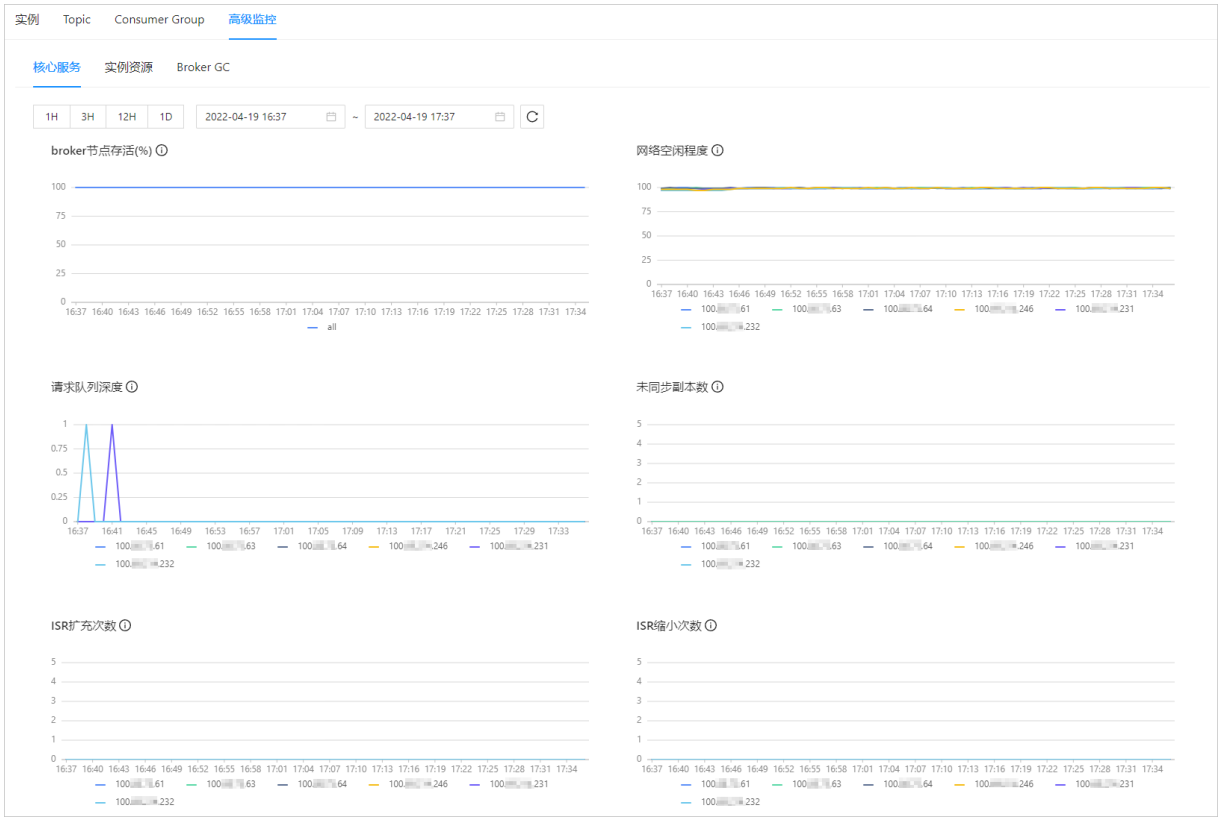
查询指定单元的监控数据

在 LDC 单元化架构环境下，可以自由切换单元，查询 Consumer Group 及其分区的监控数据，如下图所示。



高级监控

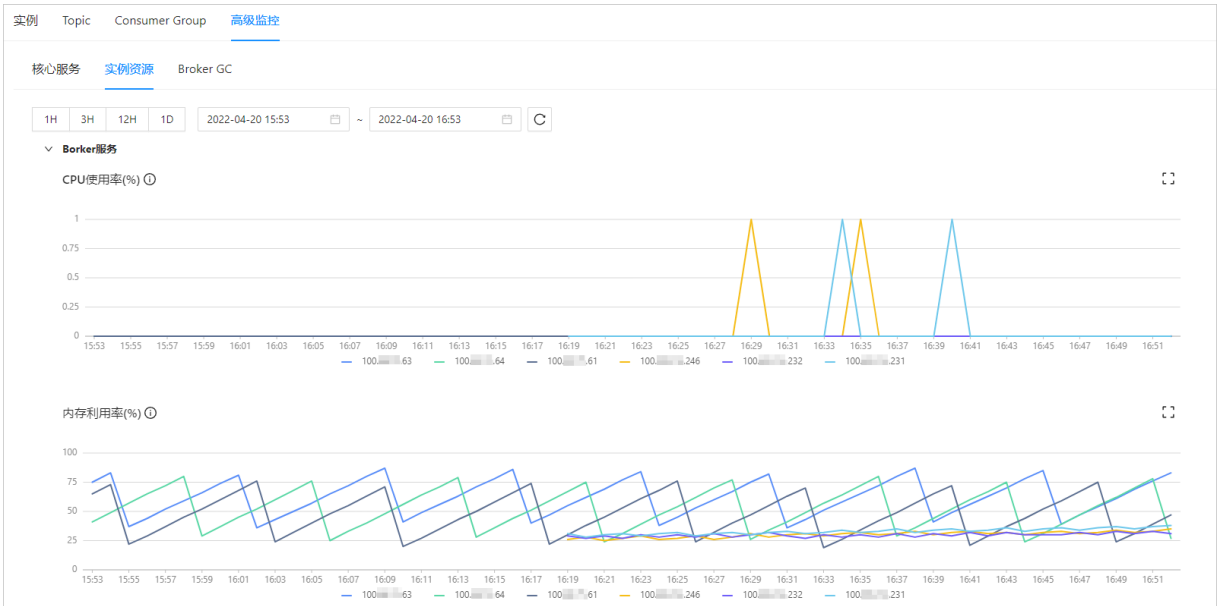
核心服务指标



参数说明如下：

参数	说明
Broker 节点存活（%）	Broker 节点存活百分比。
网络空闲程度	实例磁盘占用量（包含副本），按照所选择的时间粒度取最新值，越接近 100 代表越空闲。
请求队列深度	反映当前未处理的生产请求个数，如果该值过大可能是同一时间请求量过大，CPU 负载过高或者磁盘 IO 出现瓶颈。
未同步副本数	集群中存在的未同步的副本个数，当实例存在未同步副本，表示集群的健康度可能存在问题。
ISR 扩充次数	ISR 扩充次数，即存在未同步副本的情况下，当未同步副本追上 leader 数据，会重新加入 ISR，此时该次数就会加 1。
ISR 缩小次数	ISR 收缩次数，即当出现 Broker 宕机、Zookeeper 重连的情况，会出现 ISR 缩小的次数统计。

实例资源

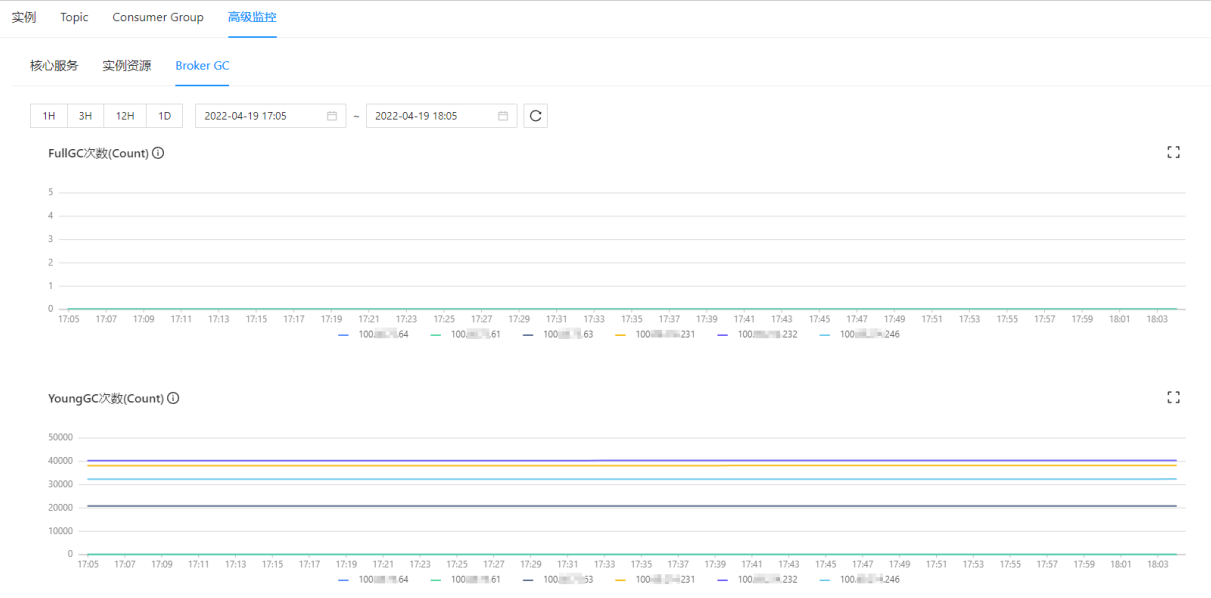


参数说明如下：

参数	说明
CPU 使用率（%）	当前 CPU 占用与实例规格 CPU 总位数的百分比。

参数	说明
内存利用率（%）	当前内存占用与实例内存总容量的百分比。

Broker GC



参数说明如下：

参数	说明
FullGC 次数（Count）	实例触发 FullGC 的次数。如果 FullGC 次数为 1，请收集并保存 JVM 相关数据，联系技术支持人员处理。
YoungGC 次数（Count）	实例触发 YoungGC 次数。如果 YoungGC 次数持续偏高，则需要调整 GC 参数。

9.ACL 策略管理

ACL 是 SOFAKafka 提供的管理 SASL 用户和客户端使用 SDK 收发消息权限的服务。您可以通过添加 ACL 策略来管理用户 Topic 资源的读写权限，以此增强用户对 Topic 资源的生产消费权限控制。

添加 ACL 策略

前提条件

已添加用户。如需新增用户，请参见 添加用户。

操作步骤

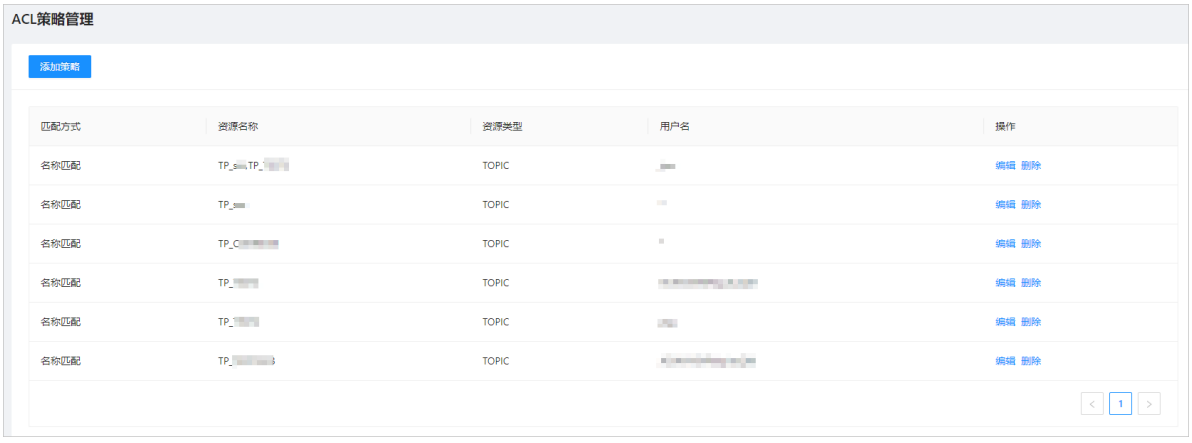
1. 登录 SOFAKafka 控制台。
2. 在左侧导航栏上，单击 **ACL 策略管理**。
3. 单击 **添加策略**，然后在 **新建 ACL 策略** 对话框配置以下信息。

参数	说明

参数	说明
匹配方式	支持选择 名称匹配 和 前缀匹配 为用户授予权限。 ○ 名称匹配：勾选需要配置相同 ACL 策略的 Topic。 ○ 前缀匹配：输入 Topic 名称前缀，按 Topic 名称前缀模糊匹配需要配置相同 ACL 策略的 Topic。
操作权限	支持选择 DENY 和 ALLOW。 ○ DENY：拒绝用户读写 Topic 资源。 ○ ALLOW：允许用户读写 Topic 资源。
用户	选择用户，不选择默认为全部用户授权。
策略	支持选择 ALL、 READ 和 WRITE。 ○ ALL：为用户添加读写的权限。 ○ READ：为用户添加读的权限。 ○ WRITE：为用户添加写的权限。

4. 单击 提交。

   授权成功后，可以在 ACL 策略管理列表查看授权的策略信息。



更多操作

- 编辑 ACL 策略：单击目标匹配方式操作列下的 编辑，在 编辑 ACL 策略 页修改相关信息，单击 提交。
- 删除策略：单击目标匹配方式操作列下的 删除，在弹出的信息提示框中，单击 是。

10.用户管理

在 SOFAKafka 实际使用过程中，为了控制消息收发权限，需要将团队成员添加至 SOFAKafka 中，然后给成员配置不同的权限去管理 Topic 资源。本文将介绍如何在 SOFAKafka 控制台中添加新成员。

添加用户

前提条件

已拥有 IAM 账号，并获得 IAM 控制台在 ROC 租户下的个人账户的账号及密码。

操作步骤

1. 登录 SOFAKafka 控制台。
2. 在左侧导航栏上，单击 **用户管理**。
3. 单击 **新增**，然后在 **新增用户** 对话框配置以下信息。

   - **用户名**：填写 IAM 在 ROC 租户下的个人账户的账号。
   - **密码**：填写 IAM 在 ROC 租户下的个人账户的密码。
   - **确认密码**：确认密码。
4. 单击 **提交**。

   您可以在 **用户管理** 页面查看刚创建的用户，此时用户还没有任何资源的操作权限，您需要对用户进行授权，具体操作说明详见 添加 ACL 策略。

更多操作

- 修改用户密码：单击目标用户操作列下的 **编辑**，在弹出的信息提示框中修改密码，单击 **提交**。
- 删除用户：单击目标用户操作列下的 **删除**，在弹出的信息提示框中，单击 **是**。

11.Java SDK 参考

11.1. 准备环境

在运行收发消息的 Java 代码前，您需按照本文提供的步骤来准备环境。

前提条件

- 安装 1.8 或以上版本 JDK
- 安装 2.5 或以上版本 Maven

操作步骤

准备配置

1. 在 `pom.xml` 中添加以下依赖。

```
<dependency>
 <groupId>com.alipay.sofa</groupId>
 <artifactId>sofakafka-client</artifactId>
 <version>1.0.3</version>
</dependency>
```

2. 创建 SOFAKafka 配置文件 `kafka.properties.java` 。

```
## 配置接入点。您可以在控制台的概览页面查看接入点。
bootstrap.servers=xxxxxxxxxxxxxxxx
## 配置 Topic。您可以在控制台的 Topic 管理页面创建 Topic。
topic=XXX
## 配置 Consumer Group。SOFAKafka 会为您自动创建该 Consumer Group。
group.id=XXX
```

11.2. 参数说明

SOFAKafka 提供 Java SDK 实现消息发送与订阅，您可通过本文了解消息发送和订阅相关的参数说明。

通用参数

参数名	说明
ENDPOINT	设置 TCP 协议接入点。登录 SOFAKafka 控制台，在 概览 > 接入配置 页面获取。
INSTANCE_ID	设置实例 ID。登录 SOFAKafka 控制台，在 概览 > 接入配置 页面获取。
ACCESS_KEY	您在 IAM 管理控制台中创建的 AccessKey，用于身份认证。
SECRET_KEY	您在 IAM 管理控制台中创建的 AccessKeySecret，用于身份认证。

LDC 参数

参数名	说明
LDC	设置为 `true`，表示为 LDC 场景。
CELL	设置为具体的 ZONE 值，表示向某个 ZONE 内的 Topic 发送消息。例如：`RZ00A`。

SASL 参数

参数名	说明
SECURITY_PROTOCOL_CONFIG	设置为 `SASL_PLAINTEXT`，表示开始 SASL 权限校验。
SASL_MECHANISM	设置为 `SCRAM-SHA-512`。
SASL_JAAS_CONFIG	设置为您在 SOFAKafka 控制台的 **用户管理** 页面中创建的用户账户与密码，例如 `org.apache.kafka.common.security.scram.ScramLoginModule required username='username' password='password';`。其中 `username` 和 `password` 按创建的用户账号密码填写即可。

消息发送参数

参数名	说明
KEY_SERIALIZER_CLASS_CONFIG	key 序列化配置，支持 String、Integer 等方式。
VALUE_SERIALIZER_CLASS_CONFIG	value 序列化配置，支持 String、Integer 等方式。

参数名	说明
ACKS_CONFIG	该参数指定必须有多少个分区副本收到消息，生产者才会认为消息写入成功。 • acks=0，生产者不需要等待服务器响应，即会认为消息已经发送成功，该模式可以达到很高的吞吐量，但是消息也相对容易丢失。 • acks=1（默认配置），只要集群的 Leader 接收到消息并且写入本地磁盘，无论其他 Follower 有没有同步到这条消息，生产者即会认为消息已经发送成功。 • acks=all，表示只有当所有同步副本全部收到消息后，生产者才会收到来自服务器的成功响应，该模式相对安全，但延迟相对更高。 重要 开启幂等后，该值必须为 all。
RETRIES_CONFIG	该参数值决定了生产者可以重发消息的次数，当达到这个次数仍然发送失败时，生产者会放弃重试并返回错误。 说明 事务消息必须设置 >0。
BATCH_SIZE_CONFIG	当多个消息发送到相同分区时，生产者会将消息放到同一批次中，一起发出。该参数值指定了一个批次可以使用的内存大小。
TRANSACTIONAL_ID_CONFIG（事务消息）	该参数为事务 id，当有多个生产者时，标识事务所属的生产者，可用于消息幂等。
ENABLE_IDEMPOTENCE_CONFIG（事务消息）	该参数为幂等性设置，使用事务消息需要将其设置为 `true`，设置了 `transactional.id` 属性后，`enable.idempotence` 属性会自动设置为 `true`。

消息接收参数

参数名	说明
KEY_DESERIALIZER_CLASS_CONFIG	key 反序列化配置，支持 String、Integer 等方式。

参数名	说明
VALUE_DESERIALIZER_CLASS_CONFIG	value 反序列化配置，支持 String、Integer 等方式。
GROUP_ID_CONFIG	该参数为 consumer group 的值，即消息需要在哪个消费组消费。
ISOLATION_LEVEL_CONFIG（事务消息）	该参数为提交事务的隔离级别，使用事物消息时，该值需要设为 `read_committed` ，即消费者只消费已经提交的消息。

11.3. 收发送普通消息（两种方式）

SOFAKafka 提供两种方式来发送普通消息：同步发送和异步发送。本文介绍了每种发送方式的原理以及示例代码。

发送同步消息

原理

同步发送是指消息发送方发出一条消息后，会在收到服务端返回响应之后才发下一条消息的通讯方式。

示例代码

> **说明**
>
> 其中以 `$` 开头的部分配置需要在使用时，改成当前环境的真实数据。

```
Properties props = new Properties();
//sofakafka 一些基本配置
props.put(ProducerConfig.ACKS_CONFIG, "all");
props.put(ProducerConfig.RETRIES_CONFIG, 0);
props.put(ProducerConfig.BATCH_SIZE_CONFIG, 16384);
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
//SASL 配置
props.put(AdminClientConfig.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
props.put(SaslConfigs.SASL_MECHANISM, "SCRAM-SHA-512");
//此处需要在console控制台创建对应的用户
props.put(SaslConfigs.SASL_JAAS_CONFIG, "org.apache.kafka.common.security.scram.ScramLoginModu
le required username='$username' password='$password';");
//sofaKafka config
//ldc场景需要填写以下两个参数，其中cell需要填写真实存在的cell
props.put(SofaKafkaProducerConfig.CELL_CONFIG, "$cell");
props.put(SofaKafkaProducerConfig.LDC_CONFIG, "true");

//instanceId
props.put(SofaKafkaProducerConfig.INSTANCE_ID_CONFIG, "$instanceId");
//accessKey
props.put(SofaKafkaProducerConfig.ACCESS_KEY_CONFIG, "$accessKey");
//secretKey
props.put(SofaKafkaProducerConfig.SECRET_KEY_CONFIG, "$secretKey");
props.put(SofaKafkaProducerConfig.SHARED_CONFIG, "shared");
//acvip地址
props.put(SofaKafkaProducerConfig.ENDPOINT_CONFIG, "$acvip_addr");
props.put(SofaKafkaProducerConfig.DATACENTER_CONFIG, "cafeb");
SofaKafkaProducer producer = new SofaKafkaProducer(props);
String topic = "$topic";

Integer count = 0;
try {
   while (true) {
      //发送消息，拿到响应结果
      Future<RecordMetadata> sendResult = producer.send(new ProducerRecord(topic, "搜索" + count.t
oString()));
      RecordMetadata recordMetadata = sendResult.get();
      System.out.println(String.format("send msg: %s,offset: %s,partition: %s", count.toString(),record
Metadata.offset(),recordMetadata.partition()));
      count++;
      Thread.sleep(1000);
   }
} catch (Exception e) {
   System.out.println(e.getMessage());
} finally {
   producer.close();
}
```

发送异步消息

原理

异步发送是指发送方发出一条消息后，不等服务端返回响应，接着发送下一条消息的通讯方式。

消息队列的异步发送，需要用户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，通过回调接口接收服务端响应，并处理响应结果，无需等待服务端响应即可发送第二条消息。

示例代码

> **说明**
>
> 其中以 `$` 开头的部分配置需要在使用时，改成当前环境的真实数据。

```
Properties props = new Properties();
//sofakafka 一些基本配置
props.put(ProducerConfig.ACKS_CONFIG, "all");
props.put(ProducerConfig.RETRIES_CONFIG, 0);
props.put(ProducerConfig.BATCH_SIZE_CONFIG, 16384);
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
//SASL 配置
props.put(AdminClientConfig.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
props.put(SaslConfigs.SASL_MECHANISM, "SCRAM-SHA-512");
//此处需要在console控制台创建对应的用户
props.put(SaslConfigs.SASL_JAAS_CONFIG, "org.apache.kafka.common.security.scram.ScramLoginModu
le required username='$username' password='$password';");
//sofaKafka config
//ldc场景需要填写以下两个参数，其中cell需要填写真实存在的cell
props.put(SofaKafkaProducerConfig.CELL_CONFIG, "$cell");
props.put(SofaKafkaProducerConfig.LDC_CONFIG, "true");

//instanceId
props.put(SofaKafkaProducerConfig.INSTANCE_ID_CONFIG, "$instanceId");
//accessKey
props.put(SofaKafkaProducerConfig.ACCESS_KEY_CONFIG, "$accessKey");
//secretKey
props.put(SofaKafkaProducerConfig.SECRET_KEY_CONFIG, "$secretKey");
props.put(SofaKafkaProducerConfig.SHARED_CONFIG, "shared");
//acvip地址
props.put(SofaKafkaProducerConfig.ENDPOINT_CONFIG, "$acvip_addr");
props.put(SofaKafkaProducerConfig.DATACENTER_CONFIG, "cafeb");
SofaKafkaProducer producer = new SofaKafkaProducer(props);
String topic = "$topic";

Integer count = 0;
try {
   while (true) {
      //发送消息，拿到响应结果
      Future<RecordMetadata> send = producer.send(new ProducerRecord("TP_jmc", "搜索" + count.toSt
ring()), new Callback() {
         @Override
         public void onCompletion(RecordMetadata recordMetadata, Exception e) {
            if (e == null) {
               System.out.println(String.format("=======PRODUCER_ASYNE========:topic:%s, partition:
%s, offset:%s", recordMetadata.topic(), recordMetadata.partition(), recordMetadata.offset()));
            }
```

```
            }
        });
        System.out.println(String.format("send msg: %s,offset: %s,partition: %s", count.toString()));
        count++;
        Thread.sleep(1000);
    }
} catch (Exception e) {
    System.out.println(e.getMessage());
} finally {
    producer.close();
}
```

订阅消息

示例代码如下：

```
Properties props = new Properties();
props.put(SofaKafkaConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getN
ame());
props.put(SofaKafkaConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.ge
tName());
props.put(SofaKafkaConsumerConfig.GROUP_ID_CONFIG, "$groupId");
//SASL 配置
props.put(AdminClientConfig.SECURITY_PROTOCOL_CONFIG,"SASL_PLAINTEXT");
props.put(SaslConfigs.SASL_MECHANISM,"SCRAM-SHA-512");
props.put(SaslConfigs.SASL_JAAS_CONFIG,"org.apache.kafka.common.security.scram.ScramLoginModul
e required username='$username' password='$password';");
//ldc场景需要填写以下两个参数，其中cell需要填写真实存在的cell
props.put(SofaKafkaProducerConfig.CELL_CONFIG, "$cell");
props.put(SofaKafkaProducerConfig.LDC_CONFIG, "true");
//instanceId
props.put(SofaKafkaProducerConfig.INSTANCE_ID_CONFIG, "$instanceId");
//accessKey
props.put(SofaKafkaProducerConfig.ACCESS_KEY_CONFIG, "$accessKey");
//secretKey
props.put(SofaKafkaProducerConfig.SECRET_KEY_CONFIG, "$secretKey");
props.put(SofaKafkaProducerConfig.SHARED_CONFIG, "shared");
//acvip地址
props.put(SofaKafkaProducerConfig.ENDPOINT_CONFIG, "$acvip_addr");
props.put(SofaKafkaProducerConfig.DATACENTER_CONFIG, "cafeb");
SofaKafkaConsumer consumer = new SofaKafkaConsumer(props);
List<String> topic = new ArrayList<>();
topic.add("$topic");
consumer.subscribe(topic);
try {
    while (true) {
        ConsumerRecords<String, String> records = consumer.poll(10);
        for (ConsumerRecord<String, String> record : records) {
            System.out.println(String.format("partition:%s, key:%s, value:%s, offset:%s", record.partition(),
record.key(), record.value(), record.offset()));
        }
        Thread.sleep(100);
    }
} catch (Exception e) {
    e.printStackTrace();
} finally {
    System.out.println("error close");
    consumer.close();
}
```

11.4. 收发事务消息

发送普通事务消息

原理

事务特性是指一系列的生产者生产消息和消费者提交偏移量的操作在一个事务中，或者说是一个原子操作，生产消息和提交偏移量同时成功或者失败。即一个事务中的多条消息，要么都发送成功，要么都发送失败。

示例代码

```
Properties props = new Properties();
//sofakafka 一些基本配置
props.put(ProducerConfig.ACKS_CONFIG, "all");
props.put(ProducerConfig.BATCH_SIZE_CONFIG, 16384);
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
//SASL 配置
props.put(AdminClientConfig.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
props.put(SaslConfigs.SASL_MECHANISM, "SCRAM-SHA-512");
//此处需要在console控制台创建对应的用户
props.put(SaslConfigs.SASL_JAAS_CONFIG, "org.apache.kafka.common.security.scram.ScramLoginModu
le required username='$username' password='$password';");
//sofaKafka config
//ldc场景需要填写以下两个参数，其中cell需要填写真实存在的cell
props.put(SofaKafkaProducerConfig.CELL_CONFIG, "$cell");
props.put(SofaKafkaProducerConfig.LDC_CONFIG, "true");

//事务消息配置，transactional id
props.put(ProducerConfig.TRANSACTIONAL_ID_CONFIG,"$transactionId");
props.put(SofaKafkaProducerConfig.ENABLE_IDEMPOTENCE_CONFIG,true);
//开启幂等producer后还需设置重试次数大于0 不然会报错
props.put(ProducerConfig.RETRIES_CONFIG, 3);

//instanceId
props.put(SofaKafkaProducerConfig.INSTANCE_ID_CONFIG, "$instanceId");
//accessKey
props.put(SofaKafkaProducerConfig.ACCESS_KEY_CONFIG, "$accessKey");
//secretKey
props.put(SofaKafkaProducerConfig.SECRET_KEY_CONFIG, "$secretKey");
props.put(SofaKafkaProducerConfig.SHARED_CONFIG, "shared");
//acvip地址
props.put(SofaKafkaProducerConfig.ENDPOINT_CONFIG, "$acvip_addr");
props.put(SofaKafkaProducerConfig.DATACENTER_CONFIG, "cafeb");
SofaKafkaProducer producer = new SofaKafkaProducer(props);
String topic = "$topic";

//初始化事务
producer.initTransactions();
Integer count = 0;
try {
   //开启事务
   producer.beginTransaction();
   while (true) {
      //发送消息，拿到响应结果
      Future<RecordMetadata> sendResult = producer.send(new ProducerRecord(topic, "搜索" + count.t
oString()));
      RecordMetadata recordMetadata = sendResult.get();
      System.out.println(String.format("send msg: %s,offset: %s,partition: %s", count.toString(),record
Metadata.offset(),recordMetadata.partition()));
      count++;
      Thread.sleep(1000);
   }
```

```
    //提交事务
    producer.commitTransaction();
} catch (Exception e) {
    System.out.println(e.getMessage());
    //终止事务
    producer.abortTransaction();
} finally {
    producer.close();
}
```

发送 Consume-transform-produce 模式事务消息

原理

该模式是在一个事务中，既有生产消息操作又有消费消息操作，但是事务回滚时，由于消费端消费过的数据无法再次被消费，所以消费过的数据不会一起回滚。

示例代码

```
// 创建生产者
SofaKafkaProducer producer = getProducer();
// 创建消费者
SofaKafkaConsumer consumer = getConsumer();
// 初始化事务
producer.initTransactions();
// 订阅主题
consumer.subscribe(Arrays.asList("$topic"));
int count= 0 ;
while (true){
   count++;
   // 开启事务
   producer.beginTransaction();
   // 接受消息
   ConsumerRecords<String, String> records = consumer.poll(10);
   for (ConsumerRecord<String, String> record : records) {
      System.out.println(String.format("partition:%s, key:%s, value:%s, offset:%s", record.partition(), r
ecord.key(), record.value(), record.offset()));
   }
   // 处理逻辑
   try {
      Map<TopicPartition, OffsetAndMetadata> commits = new HashMap<>();
      for(ConsumerRecord record : records){
         // 处理消息
         System.out.printf("offset = %d, key = %s, value = %s\n", record.offset(), record.key(), record.val
ue());
         // 记录提交的偏移量
         commits.put(new TopicPartition(record.topic(),record.partition()),new OffsetAndMetadata(reco
rd.offset()));
         // 产生新消息
         Future<RecordMetadata> metadataFuture = producer.send(new ProducerRecord<>("$topic",rec
ord.value()+"send"));
         /*if (count > 2){
            int a = 1/0;
         }*/
      }
      // 提交偏移量
      producer.sendOffsetsToTransaction(commits,"$group");
      // 事务提交
      producer.commitTransaction();

   }catch (Exception e){
      e.printStackTrace();
      producer.abortTransaction();
   }
}
```

订阅事务消息

订阅事物消息与订阅普通消息基本一致，可以参考 订阅消息，但是需要在普通订阅消息的基础上，再加上一个配置： `props.put(SofaKafkaConsumerConfig.ISOLATION_LEVEL_CONFIG,"read_committed");` ，该配置表示，消费者只订阅已提交的消息。

11.5. 收发本地优先消息

在 SOFAKafka 多机房环境下，开启本地优先后，可以发送本地优先消息。

原理

本地优先模式下，消息会优先发往本地机房的 Broker，优先被本地机房的消费者消费，只有本地机房没有 Broker/消费者时，才会发往另一个机房的 Broker/消费者。

示例代码

发送本地优先消息

```
Properties props = new Properties();
//sofakafka 一些基本配置
props.put(ProducerConfig.ACKS_CONFIG, "all");
props.put(SofaKafkaProducerConfig.RETRIES_CONFIG, 1);
props.put(ProducerConfig.BATCH_SIZE_CONFIG, 16384);
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
//SASL 配置
props.put(AdminClientConfig.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
props.put(SaslConfigs.SASL_MECHANISM, "SCRAM-SHA-512");
//此处需要在console控制台创建对应的用户
props.put(SaslConfigs.SASL_JAAS_CONFIG, "org.apache.kafka.common.security.scram.ScramLoginModu
le required username='$username' password='$password';");
//sofaKafka config
//instanceId
props.put(SofaKafkaProducerConfig.INSTANCE_ID_CONFIG, "$instanceId");
//accessKey
props.put(SofaKafkaProducerConfig.ACCESS_KEY_CONFIG, "$accessKey");
//secretKey
props.put(SofaKafkaProducerConfig.SECRET_KEY_CONFIG, "$secretKey");
props.put(SofaKafkaProducerConfig.SHARED_CONFIG, "shared");

//本地优先配置
props.put(SofaKafkaProducerConfig.LDC_CONFIG, "false");
//开启本地优先
props.put(SofaKafkaProducerConfig.LOCAL_FIRST_ENABLE_CONFIG, true);
//此处填写互为容灾的多机房的datacenter信息即可
props.put(SofaKafkaConsumerConfig.LOCAL_FIRST_DATACENTERS_CONFIG, "$A-Datacenter,$B-Datace
nter");
//填写本地机房的acvip地址即可
props.put(SofaKafkaProducerConfig.ENDPOINT_CONFIG, "$acvip_addr");
//此处填写本地机房的datacenter地址配置，例如本地机房为a机房，则应为$A-Datacenter
props.put(SofaKafkaProducerConfig.DATACENTER_CONFIG, "$A-Datacenter");
SofaKafkaProducer producer = new SofaKafkaProducer(props);
Integer count = 0;
try {
    Random random = new Random();
    while (true) {//循环1 ~ 3秒发一次消息
        String msg = "msg, value" + count;
        ProducerRecord sofaKafkaProducerRecord = new ProducerRecord("$topic", msg);
        producer.send(sofaKafkaProducerRecord);
        System.out.println(String.format("send msg: %s", msg));
        count++;
        Thread.sleep((random.nextInt(3)+1) * 1000);
    }
} catch (Exception e) {
    log.info(e.getMessage());
} finally {
    producer.close();
}
```

订阅本地优先消息

```
Properties props = new Properties();
//sofakafka 一些基本配置
props.put(SofaKafkaConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true");
props.put(SofaKafkaConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
//SASL 配置
props.put(AdminClientConfig.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
props.put(SaslConfigs.SASL_MECHANISM, "SCRAM-SHA-512");
//此处需要在console控制台创建对应的用户
props.put(SaslConfigs.SASL_JAAS_CONFIG, "org.apache.kafka.common.security.scram.ScramLoginModu
le required username='$username' password='$password';");
//sofaKafka config
//instanceId
props.put(SofaKafkaProducerConfig.INSTANCE_ID_CONFIG, "$instanceId");
//accessKey
props.put(SofaKafkaProducerConfig.ACCESS_KEY_CONFIG, "$accessKey");
//secretKey
props.put(SofaKafkaProducerConfig.SECRET_KEY_CONFIG, "$secretKey");
props.put(SofaKafkaProducerConfig.SHARED_CONFIG, "shared");

//本地优先配置
props.put(SofaKafkaProducerConfig.LDC_CONFIG, "false");
//开启本地优先
props.put(SofaKafkaProducerConfig.LOCAL_FIRST_ENABLE_CONFIG, true);
//此处填写互为容灾的多机房的datacenter信息即可
props.put(SofaKafkaConsumerConfig.LOCAL_FIRST_DATACENTERS_CONFIG, "$A-Datacenter,$B-Datace
nter");
//填写本地机房的acvip地址即可
props.put(SofaKafkaProducerConfig.ENDPOINT_CONFIG, "$acvip_addr");
//此处填写本地机房的datacenter地址配置，例如本地机房为a机房，则应为$A-Datacenter
props.put(SofaKafkaProducerConfig.DATACENTER_CONFIG, "$A-Datacenter");

props.put(SofaKafkaConsumerConfig.GROUP_ID_CONFIG, "$group");
SofaKafkaProducer producer = new SofaKafkaProducer(props);
consumer.subscribe(Arrays.asList("$topic"));//订阅的topic
try {
   while (true) {
      log.info("in consuming...");
      ConsumerRecords<String, String> records = consumer.poll(10);
      for (ConsumerRecord<String, String> record : records) {
         log.info(String.format("partition:%s, offset:%s, key:%s, value:%s", record.partition(), record.off
set(), record.key(), record.value()));
      }
      Thread.sleep(1000);
   }
} catch (Exception e) {
   log.error(e.getMessage());
} finally {
   log.info("error close");
   consumer.close();
}
```