消息队列 CKafka 版

常见问题

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

常见问题
实例问题

最近更新时间：2025-07-18 11:13:43

如何选择实例的规格?

可以借助控制台的 "迁移上云 – 规格计算器" 来计算所需的产品规格：

消息队列 CKafka 版实例按照规格分为标准版和专业版，两个版本的对比参见 产品规格 。
请根据自身需要选择产品规格，不同规格的计费规则请参见 计费概述 。

实例升配有哪些模式? 主要有什么区别?

标准版和专业版的多种迁移模式，都是无损的，不停服，客户端无感知的。
标准版是共享集群，当集群的空余资源足够的时候，升配模式是以提高配额的形式进行的。不会涉及到数据迁移，故正常几分钟内就升配完成。如果集群空余资源不足，就会提示资源不足，需要 提工单 进行后台迁移并调配资源，然后再在控制台进行正常的标准版升配流程。
专业版是独占的集群，当系统算法计算到当前独占的资源池不够升配后的资源配额的时候，需要自动化的添加资源到资源池，并对实例的数据进行迁移。故专业版升配存在两种模式：

- 资源足够时，即时升配提高配额的模式，几分钟内升配完成。
- 资源不足时，数据迁移的模式，视集群堆积的数据大小而定，可能需要几分钟到几个小时。

当需要数据迁移时，可自定义迁移时间：

当需要数据迁移时，会有迁移模式和定时迁移两种选项，迁移模式分为如下两种：

- 稳定模式：CKafka 将限制升配过程中数据迁移速度，最大程度保留实例的带宽属性，适合于不希望干扰业务的场景。
- 高速模式：CKafka 将不对升配过程中数据迁移的速度进行限制，会影响实例的生产消费带宽，适合于业务低峰或者允许停服的场景。

因为迁移需要占用集群的带宽和磁盘资源，在业务高峰的时候，客户可能担心迁移会影响业务。故可以设置定时迁移，例如设置到半夜进行升配。升配操作详情请参见 升配实例 。

为什么集群带宽百分比未超过100%，却产生了限流?

从监控界面中，可以看到集群实例的生产或消费带宽百分比未超过100%，但实例最大生产或消费带宽已经超出了所购买集群的带宽，同时集群限流次数不为 0。出现这种现象有以下两种原因：

1. 目前集群所使用的限流机制，针对 Kafka 这样大流量的场景下，采用了较为柔和的延时回包机制。所以在极端情况下，实例最大生产/消费带宽可以超出所购集群的带宽。详情可以参考限流机制说明 。
2. 带宽百分比指标，采用较为削峰填谷的方式呈现当前集群的使用比例，我们的计算方式为获取单位时间内的平均值。故而出现上述的问题。

实例最大生产/消费带宽已经超出了所购买集群的带宽，但未产生限流?

目前集群所使用的限流机制，针对 Kafka 这样大流量的场景下，采用了较为柔和的延时回包机制。所以在极端情况下，实例最大生产/消费带宽可以超出所购集群的带宽。详情可以参见 限流机制说明 。

实例是否支持数据压缩?

当前 CKafka 支持开源的 snappy 和 lz4 的消息压缩格式。由于 Gzip 压缩对于 CPU 的消耗较高，暂未支持。开启压缩在某些情况下可能会加大服务端的压力，例如生产者的版本较高，消费者的版本协议较低，在消费的时候，会出现向下的协议转换。导致服务端压力升高。所以建议如果需要压测，建议客户关闭消息压缩参数进行测试。
配置开启方法：Producer 的配置文件中参数 compression.type = snappy 或者 lz4，默认为关闭 none。
建议优先选择 Snappy 压缩。开启压缩后，可能会导致服务端 CPU 升高，导致生产消费耗时变高。需谨慎使用。

Topic 问题

最近更新时间：2025-03-17 14:34:24

如何选取 CKafka 副本数?

建议您创建 Topic 时选择2副本或3副本存储数据，保障数据可靠性。默认创建的 Topic 是2副本，如果业务需要更高的可用性，则可以指定选择3副本运行。如果 Topic 需要更多的副本数，可以 提交工单 进行处理。
为了提高数据的安全性，CKafka 不建议创建单副本的 Topic，如您账户下有单副本的 Topic，建议按如下步骤迁移：

1. 创建新的 Topic，选择相同的分区，选取双副本。
2. 生产消息到新的 Topic 中，存量的单副本 Topic 继续被消费。
3. 消费完毕后修改消费者配置，订阅新的 Topic 进行消费。

为什么设置了 Topic 消息保留的时间，在其设置的时间点之后仍然可以查询到消息?

1. 消息中增加了一个时间戳字段和时间戳类型。目前支持的时间戳类型有两种：CreateTime 和 LogAppendTime 前者表示生产者创建这条消息的时间； 后者表示broker接收到这条消息的时间。如果客户端生产消息时间的时间戳数据不合法，会影响 broker 服务端删除数据。
2. 主题中分区数过多，消息数据过少，且分区内只存在一个日志段文件，则不会删除消息。
3. 日志删除任务会检查当前日志的大小是否超过设定的阈值, 即每个分段1GB大小。若日志段中最大的时间戳数据在保留的时间内则不会删除消息。

为什么 Topic 数量（分区总数）存在限制?

Kafka 的 Topic 总数（分区总数）太多会使集群性能和稳定性下降。
因为单机可承载的分区数是有限度的，如果需要更多的分区那么就需要添加节点，需要更多的成本投入。Kafka的存储和协调机制是以分区为粒度的，分区数太多，会导致存储碎片化严重，单机层面的随机写增多，集群层面分区 Leader 切换效率降低，Controller 节点故障切换变慢等情况。导致集群性能和稳定性下降。

Topic 数量和分区数量有什么关系?

CKafka 以分区（partition）作为分配单位。
总分区数 = TopicA × 副本数 + TopicB × 副本数 + ... + TopicN × 副本数。
即副本数也算在总分区个数里面，例如客户创建了1个 Topic、6个分区、2个副本，那么分区额度一共用了1 × 6 × 2 = 12个。

- 分区数：一个物理上分区的概念，一个 Topic 可以包含一个或者多个 partition。
- 副本数：partition 的副本个数，用于保障 partition 的高可用，为保障数据可靠性，当前不支持创建单副本 Topic，默认开启2副本。

Consumer Group 问题

最近更新时间：2025-03-17 14:34:24

1. 怎样设置合理的消费者数量?

消费组和消费者的对应关系如下：

- 一个消费者可以同时订阅多个 Topic。
- 一个 Topic 里面包含了1到多个分区。
- 一个分区只能被一个消费者消费。

所以，一个消费组里面，消费者数量上限 = topic1的分区数 + topic2的分区数 +..... + topicN 的分区数。
关于消费者的定义：消费者在代码层面指的是一个 Consumer 的对象，一个机器上可以有多个消费者，例如起多个线程，一个线程里面有一个 Consumer，以此类推，如下面代码所示：

```
Properties props = new Properties();
props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrap);
KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList(topic));
while (true) {
            ConsumerRecords<String, String> records =
consumer.poll(100);
}
```

初始的消费者数量可以根据客户端的资源情况进行初始的部署，然后配置消费者组的堆积告警，如果出现堆积的时候再扩容消费者。配置告警方式参见 配置告警 。

2. 为什么我看不到消费者详情?

消费者详情，包括消费者消费的 Topic、分区进度以及客户端等信息。这些信息是消费者处理完成消息，向服务端提交 offset 后，服务端进行存储的。以下情况客户端不会自动往服务端提交 offset 存储请求：

1. 使用自定义分区消费的模式；
2. 使用 Flink + CKafka 实现大数据场景，Flink 集群消费由于不提交 offset，也不通过 coordinator 进行 partition rebalance，所以在服务端没有对应的消费状态以及消费 pod IP 信息。

如果排除上述场景后，可以参见 Consumer Group 列表详情缺失 进行问题排查。

客户端问题

最近更新时间：2025-05-09 10:14:12

通用问题

Kafka Console 客户端测试时看不到数据如何处理?

- 消费者采用 latest 时候只会获取最后的数据，需要同时保持生产才可以看到相应数据。
- 改为 earliest 方式消费数据。

新接入客户端时生产或消费错误?

- 检查 telnet 是否联通（网络问题，是否 Kafka 和生产者在相同网络环境下）。
- 访问的 vip - port 是否配置正确。
- topic 白名单是否开启，如果开启需要配置正确的 IP 才能访问。

客户端生产消息如何保证在同一分区是有序的?

- 如果 Topic 只有一个分区，那么消息会根据服务端收到的数据顺序存储，则数据就是分区有序的。
- 如果 Topic 有多个分区，可以在生产端指定这一类消息的 key，这类消息都用相同的 key 进行消息发送，CKafka 会根据 key 哈希取模选取其中一个分区进行存储，由于一个分区只能由一个消费者进行监听消费，此时消息就具有消息消费的顺序性了。

对于单个生产者来说，对单个分区的生产，是保持有序的。

客户端生产消息一般会与 Broker 建立多少个连接?

从单个客户端实例（new 一个 Producer 对象的角度）来看，和所有服务端建立的连接总数公式如下：总连接数 = 1~n（n 是 Broker 的数量）。
每个 Java Producer 端管理 TCP 连接的方式如下：

1. KafkaProducer 实例创建时启动 Sender 线程，从而创建与 bootstrap.servers 中所有 Broker 的 TCP 连接。
2. KafkaProducer 实例首次更新元数据信息之后，还会再次创建与集群中所有 Broker 的 TCP 连接。
3. 如果 Producer 端发送消息到某台 Broker 时发现没有与该 Broke r的 TCP 连接，那么也会立即创建连接。
4. 如果设置 Producer 端 connections.max.idle.ms 参数大于0，则步骤1中创建的 TCP 连接会被自动关闭；默认情况下该参数值是9分钟，即如果在9分钟内没有任何请求“流过”某个 TCP 连接，那么 Kafka 会主动帮您把该 TCP 连接关闭。如果设置该参数=-1，那么步骤1中创建的 TCP 连接将无法被关闭，从而成为“僵尸”连接。

使用客户端发送消息后，如何确定是否发送成功?

大部分客户端在发送之后，会返回 Callback 或者 Future，如果回调成功，则说明消息发送成功。

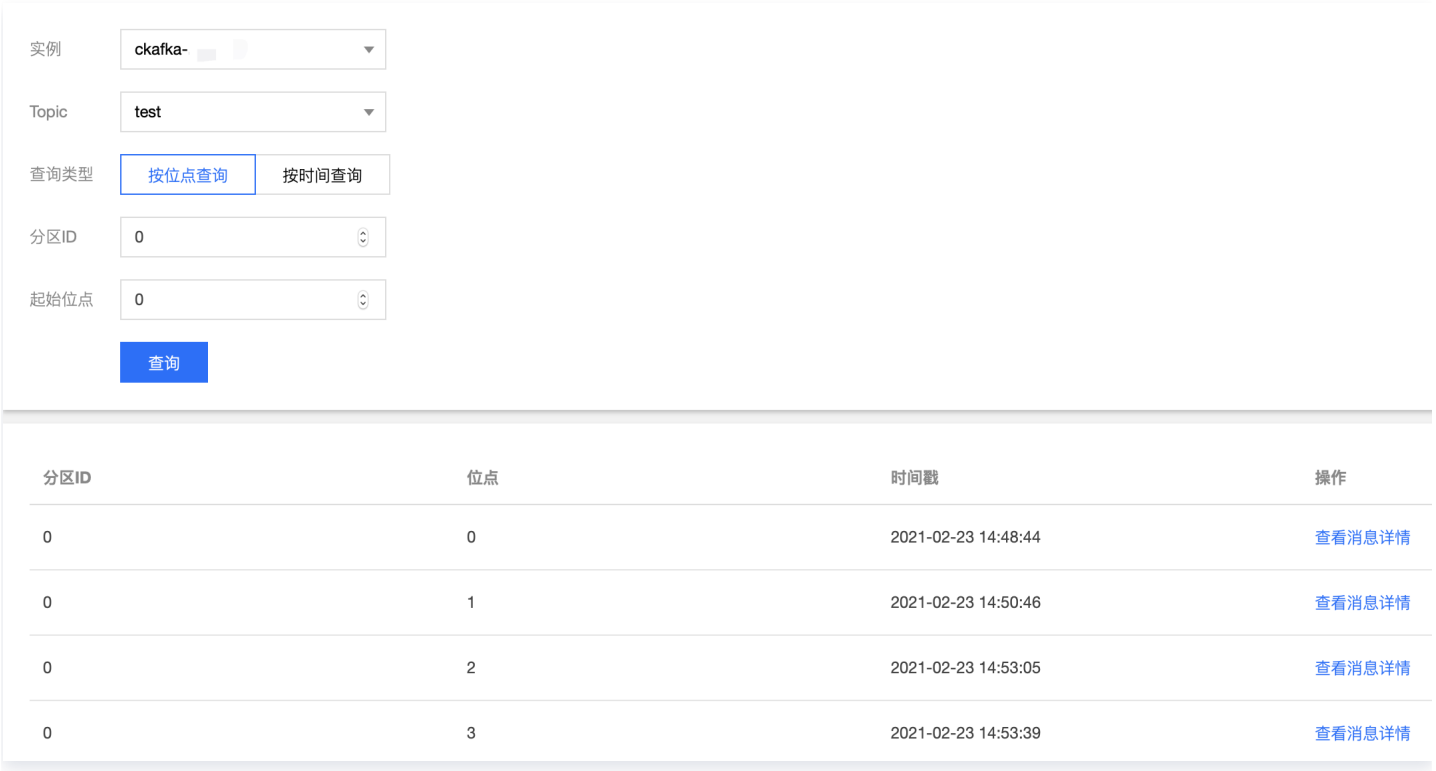
您还可以在控制台通过以下方式确认消息发送是否正常：

- 查看 Topic 的分区状态，可以实时看到各个分区的消息数量。
- 查看 Topic的 流量监控，可以看到生产消息的流量曲线。

可以通过打印 send 方法返回的 partition 和分区信息来确认是否成功：

```
Future<RecordMetadata> future = producer.send(new ProducerRecord<>
(topic, messageKey, messageStr));
RecordMetadata recordMetadata = future.get();
log.info("partition: {}", recordMetadata.partition());
log.info("offset: {}", recordMetadata.offset());
```

如果能够打印出 partition 和 offset，则表示当前发送的消息在服务端已经被正确保存。此时可以通过消息查询的工具去查询相关位点的消息即可。
如果打印不出 partition 和 offset，则表示消息没有被服务端保存，客户端需要重试。



leader 切换是什么?

在建立一个新 topic 时，kafka broker 集群会进行每个 partition 的 leader 分配，将当前 topic 的 partition 均匀分布在每个 broker 上。
但在使用一段时间后，可能会出现 partition 在 broker 上分配不均，
或是出现客户端在生产消费中抛出 `BrokerNotAvailableError` ， `NotLeaderForPartitionError` 等异

常。
这通常都是由于 partition 发生了 leader 切换导致的，典型场景如下：

- 当某个 partition leader 所在的 broker 发生某些意外情况，例如网络中断，程序崩溃，机器硬件故障导致无法与 broker controller 通信时，
  当前 topic partition 将会发生 **leader 切换**，leader 将迁移至 follower partition 上。
- 当 kafka 集群设置 `auto.leader.rebalance.enable = true` 进行自动 reBalance，或是人工增加/削减 broker 并手动触发 reBalance 时，
  由于涉及到 partition 自动平衡，此时也会出现 **leader 切换**。

当由于 broker 意外中断，导致 leader 切换时：

- 如果客户端设置 `ack = all`，并且 `min.insync.replicas > 1` ，由于消息同时在 leader partition 和 follower partition 都确认，因此消息不会丢失。
- 如果客户端设置 `ack = 1` ，此时可能会出现设置在 `replica.lag.time.max.ms` 时间中的消息未同步到 follower partition，**可能导致消息丢失**。

当由于 broker 正常，手动/自动(如实例升级、单可用区切换跨可用区、实例迁移等)发起 reBalance 导致 leader 切换时，不会导致消息丢失，原因如下：

- 如果客户端设置 `ack = all`，并且 `min.insync.replicas > 1` ，由于消息同时在 leader partition 和 follower partition 都确认，因此消息不会丢失。
- 如果客户端设置 `ack = 1` ，leader 切换将会自动同步 partition 中的 offset，因此消息不会丢失。

特定客户端已知问题

Kafka Java 客户端

Producer Partitioner 不均问题

Kafka Java Producer在2.3版本之前默认使用轮询分区策略进行消息发送，在2.4版本时候，开始提供可选的分区选择器，提供RoundRobinPartitioner和UniformStickyPartitioner两种分区策略， 默认使用UniformStickyPartitioner轮询分区策略进行分区选择。 其中RoundRobinPartitioner存在未实现的方法法onNewBatch()，可能出现部分分区进行发送的现象，从而导致不均现象，针对这类问题，可以考虑使用自定义分区器MyRoundRobinPartitioner，在初始化生产者时候指定该类即可。由于社区推荐UniformStickyPartitioner分区器作为默认分区器，但是由于Kafka开源Java SDK，版本>=2.4.1 && < 3.3.0，其中Kafka Producer默认的粘性分区策略存在数据倾斜问题，尤其linger.ms为0（短时间的的连续消息均被发送至同一个分区内），客户如果对热点数据倾斜较为敏感的场景，可以考虑Kafka Java SDK升级到3.3.0以及以上，Kafka Java SDK Producer在3.3.0开发了Strictly Uniform Sticky Partitioner对此进行优化。
高版本：写入一定要指明ack，幂等，linger.ms。

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements. See the NOTICE file distributed with
 * this work for additional information regarding copyright ownership.
```

```
 * The ASF licenses this file to You under the Apache License, Version
2.0
 * (the "License"); you may not use this file except in compliance with
 * the License. You may obtain a copy of the License at
 *
 *    http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.apache.kafka.clients.producer;

import java.util.List;
import java.util.Map;
import java.util.Queue;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentLinkedQueue;
import java.util.concurrent.ConcurrentMap;
import java.util.concurrent.atomic.AtomicInteger;

import org.apache.kafka.common.Cluster;
import org.apache.kafka.common.PartitionInfo;
import org.apache.kafka.common.utils.Utils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * The "Round-Robin" partitioner - MODIFIED TO WORK PROPERLY WITH STICKY
PARTITIONING (KIP-480)
 * <p>
 * This partitioning strategy can be used when user wants to distribute
the writes to all
 * partitions equally. This is the behaviour regardless of record key
hash.
 */
public class MyRoundRobinPartitioner implements Partitioner {
    private static final Logger LOGGER =
LoggerFactory.getLogger(RoundRobinPartitioner.class);
```

```
    private final ConcurrentMap<String, AtomicInteger> topicCounterMap =
new ConcurrentHashMap<>();
    private final ConcurrentMap<String, Queue<Integer>>
topicPartitionQueueMap = new ConcurrentHashMap<>();

    public void configure(Map<String, ?> configs) {
    }

    /**
     * Compute the partition for the given record.
     *
     * @param topic      The topic name
     * @param key        The key to partition on (or null if no key)
     * @param keyBytes   serialized key to partition on (or null if no
key)
     * @param value      The value to partition on or null
     * @param valueBytes serialized value to partition on or null
     * @param cluster    The current cluster metadata
     */
    @Override
    public int partition(
        String topic, Object key, byte[] keyBytes, Object value, byte[]
valueBytes, Cluster cluster) {
        Queue<Integer> partitionQueue =
partitionQueueComputeIfAbsent(topic);
        Integer queuedPartition = partitionQueue.poll();
        if (queuedPartition != null) {
            LOGGER.trace("Partition chosen from queue: {}",
queuedPartition);
            return queuedPartition;
        } else {
            List<PartitionInfo> partitions =
cluster.partitionsForTopic(topic);
            int numPartitions = partitions.size();
            int nextValue = nextValue(topic);
            List<PartitionInfo> availablePartitions =
cluster.availablePartitionsForTopic(topic);
            if (!availablePartitions.isEmpty()) {
                int part = Utils.toPositive(nextValue) %
availablePartitions.size();
                int partition =
availablePartitions.get(part).partition();
                LOGGER.trace("Partition chosen: {}", partition);
```

```
                return partition;
            } else {
                // no partitions are available, give a non-available
partition
                return Utils.toPositive(nextValue) % numPartitions;
            }
        }
    }

    private int nextValue(String topic) {
        AtomicInteger counter =
            topicCounterMap.computeIfAbsent(
                topic,
                k -> {
                    return new AtomicInteger(0);
                });
        return counter.getAndIncrement();
    }

    private Queue<Integer> partitionQueueComputeIfAbsent(String topic) {
        return topicPartitionQueueMap.computeIfAbsent(topic, k -> {
            return new ConcurrentLinkedQueue<>();
        });
    }

    public void close() {
    }

    /**
     * Notifies the partitioner a new batch is about to be created. When
using the sticky partitioner,
     * this method can change the chosen sticky partition for the new
batch.
     *
     * @param topic         The topic name
     * @param cluster       The current cluster metadata
     * @param prevPartition The partition previously selected for the
record that triggered a new
     *                      batch
     */
    @Override
    public void onNewBatch(String topic, Cluster cluster, int
prevPartition) {
```

```
            LOGGER.trace("New batch so enqueuing partition {} for topic {}",
prevPartition, topic);
            Queue<Integer> partitionQueue =
partitionQueueComputeIfAbsent(topic);
            partitionQueue.add(prevPartition);
        }
    }
```

Consumer coordinator 无法刷新重连

kafka java consumer客户端，版本>=2.6.x&&<3.2.1，在 commitOffsetsAsync 异常后，无法恢复与coordinator的连接，从而导致后续无法提交位点，该issue在3.2.1以及之后的版本修复，建议存在该问题的客户端，可要么升级到3.2.1进行解决，要么回滚到<2.6.x版本。

Consumer 粘性分配器重复消费问题

kafka java consumer客户端<2.3.0版本，如果使用Sticky assignor，会出现分区重复分配，即一个分区可能同时分配给同一个消费组的不同消费者消费，从而造成重复消费问题，如果使用Sticky assignor，建议客户调整版本到2.3.0以及以上，避免该问题。

Go sarama客户端

Go sarama 元数据刷新问题

1. Go sarama 客户端元数据更新参数由Metadata.Full 控制，设计实现上考虑方便默认为true，会更新全量Topic的元数据，对性能有一定影响，对于大量topic的情况下，可能影响客户使用kafka生产和消费的性能，建议客户修改该配置为false。
2. Go sarama 客户端默认的元数据刷新时间参数由Metadata.RefreshFrequency控制，默认值为10分钟，容易触及 broker 默认的连接空闲超时（10min）导致连接被中断，建议调整成5分钟与kafka java 标准sdk默认参数保持一致。

Go sarama Offset 提交过期问题

Go sarama客户端CommitOffset请求可以指定offset的过期时间，该参数由Retention.Minutes控制，默认值为0，该值会导致提交后位点会立马过期，建议设置改值与Ckafka位点过期时间保持一致，建议为7天。

Go sarama cluster 客户端的使用问题

目前 sarama cluster 已经不再维护，存在诸多缺陷，目前存在如下已知缺陷：消费者客户端底层代码 PartitionConsumer 会调用 FetchRequest 解包FetchResponse，此时 sarama cluster 实现没有按照标准协议实现，存在缺陷，即使服务端实际返回全量的元数据，但是客户端有概率出现解析到的元数据不全的问题，从而出现无法生产或者消费，报错特征为：response did not contain all the expected topic/partition blocks。

具体场景举例：在客户实例进行升降配操作，因为会增加节点，此时会触发客户端进行元数据更新，如果在元数据中，按照 test-0，a-0，test-1，a-1，test-2 的顺序返回 test 的3个分区，a的2个分区的全量元数据，虽然 broker 会返回全量的元数据，但是存在一定概率出现顺序有间隔，sarama cluster 的客户端解析时候会出现解析到不完整的元数据，比如只能解析到 a-1，test-2两个元数据，引发如上报错问题，即响应没有包含预期所有分区元数据，从而导致无法生产或者消费。
针对该情况，标准协议实现的客户端是不会出现的，例如 Java，confluent go，因此建议客户在升降配前，务必了解是否使用该客户端，因为升降配会导致元数据更新的场景十分频繁，如果有，则强烈建议客户升级为 confluent go 客户端再进行升降配，如果无法升级，在出现该问题后立刻进行客户端重启，则有概率恢复正常。

trpc-go 客户端

trpc-go kafka 库会出现消费卡住问题

trpc-go 消费端使用 kafka 库的默认配置情况下，Handle 是同步的嵌入到消息接收整个流程中的。这意味着，如果在 Handle 中做耗时非常长的任务，或者在 Handle 中返回错误，均会导致框架认为消息消费不成功，并一直重试，于是出现了不断重试同一个消息的情况，会导致消费者一直卡在当前这个消息的位置，因此，推荐客户在使用 trpc-go时候，自行处理错误重试，无论何种情况不要将任何错误返回给框架。

网络问题

最近更新时间：2024-10-14 09:57:53

CKafka 是否支持公网访问?

CKafka 默认内网传输，如需通过公网访问，需要单独开通一条公网路由，具体操作参见 添加路由策略，当前单台 Broker 默认提供3Mbps 免费公网带宽。
CKafka 专业版实例支持升配公网带宽，最高可提升至198Mbps，若您有更高的带宽需求，您可以额外支付费用购买。具体价格请参见 计费概述 。
由于公网访问会涉及延时、网络环境和安全性等问题，不建议客户长期开启公网传输。

公网开启 SASL 之后，VPC 内网如何继续访问?

如果您在开通公网访问路由的同时还使用了 PLAINTEXT 方式接入 CKafka，那么之前为 Topic 设置的 ACL 仍然会生效。若您希望 PLAINTEXT 方式的访问不受影响，请为 PLAINTEXT 需要访问的 Topic 添加全部用户的可读写的权限。

> **说明**
> 在添加 ACL 策略时，不需要选择任何用户，则默认为**全部用户**添加了读写权限。

添加完成效果如下：

监控问题

最近更新时间：2024-10-14 17:11:39

客户端生产消息如何保证在同一分区是有序的?

- 如果 Topic 只有一个分区，那么消息会根据服务端收到的数据顺序存储，则数据就是分区有序的。
- 如果 Topic 有多个分区，可以在生产端指定这一类消息的 key，这类消息都用相同的 key 进行消息发送，CKafka 会根据 key 哈希取模选取其中一个分区进行存储，由于一个分区只能由一个消费者进行监听消费，此时消息就具有消息消费的顺序性了。

对于单个生产者来说，对单个分区的生产，是保持有序的。

客户端生产消息一般会与 Broker 建立多少个连接?

从单个客户端实例（new 一个 Producer 对象的角度）来看，和所有服务端建立的连接总数公式如下：总连接数 = 1~n（n 是 Broker 的数量）。
每个 Java Producer 端管理 TCP 连接的方式如下：

1. KafkaProducer 实例创建时启动 Sender 线程，从而创建与 bootstrap.servers 中所有 Broker 的 TCP 连接。
2. KafkaProducer 实例首次更新元数据信息之后，还会再次创建与集群中所有 Broker 的 TCP 连接。
3. 如果 Producer 端发送消息到某台 Broker 时发现没有与该 Broke r的 TCP 连接，那么也会立即创建连接。
4. 如果设置 Producer 端 connections.max.idle.ms 参数大于0，则步骤1中创建的 TCP 连接会被自动关闭；默认情况下该参数值是9分钟，即如果在9分钟内没有任何请求“流过”某个 TCP 连接，那么 Kafka 会主动帮您把该 TCP 连接关闭。如果设置该参数=-1，那么步骤1中创建的 TCP 连接将无法被关闭，从而成为“僵尸”连接。

使用客户端发送消息后，如何确定是否发送成功?

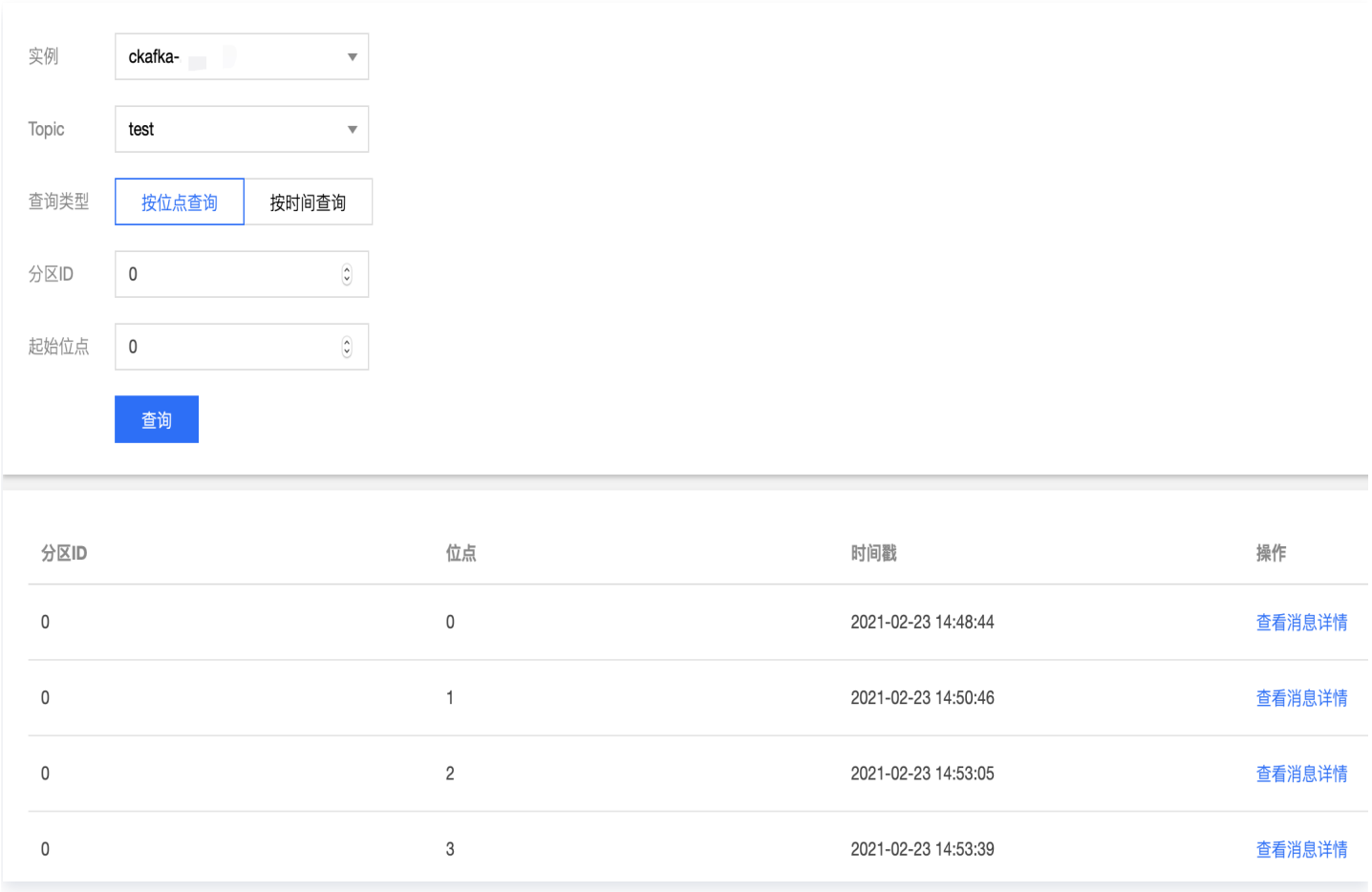
大部分客户端在发送之后，会返回 Callback 或者 Future，如果回调成功，则说明消息发送成功。
您还可以在控制台通过以下方式确认消息发送是否正常：

- 查看 Topic 的分区状态，可以实时看到各个分区的消息数量。
- 查看 Topic的 流量监控，可以看到生产消息的流量曲线。

可以通过打印 send 方法返回的 partition 和分区信息来确认是否成功：

```
Future<RecordMetadata> future = producer.send(new ProducerRecord<>
(topic, messageKey, messageStr));
RecordMetadata recordMetadata = future.get();
log.info("partition: {}", recordMetadata.partition());
log.info("offset: {}", recordMetadata.offset());
```

如果能够打印出 partition 和 offset，则表示当前发送的消息在服务端已经被正确保存。此时可以通过消息查询的工具去查询相关位点的消息即可。
如果打印不出 partition 和 offset，则表示消息没有被服务端保存，客户端需要重试。



消息问题

最近更新时间：2024-10-14 09:57:33

剩余的未消费消息的条数是如何计算的?

计算方式为：未消费消息数量 = 最大的 offset - 提交的 offset。如下图：

是否支持自动调整消息保留时间?

CKafka 支持添加动态消息保留策略功能，设置数据动态保留策略后，当磁盘空间使用率到达一定的比例后，会自动向前过期一定比例的数据，避免遇到用户消息猛增的情况，磁盘空间满了之后，则无法正常生产和消费。具体操作方式参见 添加动态消息保留策略 。

概念问题

最近更新时间：2025-03-17 14:34:24

CKafka 兼容哪个版本的开源 Kafka?

目前 CKafka 服务可以兼容0.9、0.10、1.1、2.4、2.8、3.2版本的开源 Kafka API，实现用户零成本上云。

当前的 CKafka 是基于开源 Kafka 的哪个版本?

当前 CKafka 基于 Apache Kafka 2.4、2.8、3.2版本，推荐生产消费端选取对应版本的 SDK。

消息队列 CKafka 是否会暴露 ZooKeeper?

不开放 ZooKeeper，不提供 zk 地址。

CKafka 是否支持公网访问?

当前 CKafka 默认内网传输，由于公网访问会涉及延时、网络环境和安全性等问题，不建议客户长期开启公网传输。
如果有临时公网传输需求建议联系客户经理评估。

CKafka 是否支持消息压缩?

当前 CKafka 支持开源的 snappy 和 lz4 的消息压缩格式。由于 Gzip 压缩对于 CPU 的消耗较高，暂未支持。
测试期间建议客户关闭消息压缩参数进行测试。
配置开启方法：Producer 的配置文件中参数 compression.type = snappy 或者 lz4，默认为关闭 none。

Kafka 客户端是否可以直接连接 CKafka 服务?

CKafka 可以兼容0.9、0.10、1.1、2.4、2.8、3.2版本的开源 Kafka，您可以通过 Kafka 客户端连接消息中心，并且把代码部署在腾讯云服务中生产或消费消息。

CKafka 实例有哪些限制?

根据实例的不同规格，对峰值吞吐量、磁盘容量、实例级别 Topic 数、实例级别 partition 有不同限制，具体可参见 计费概述 。

CKafka 是否会丢失消息?

- 开源的 Apache Kafka 不保证不丢消息；CKafka 针对可用性做了优化，腾讯云承诺 CKafka 的可用性超99.95%。
- CKafka 客户可以通过生产时开启 ACK ，尽量保障不丢失和少丢失消息，提升消息可靠性。
- 变更集群或升级过程对客户透明，秒级变更。

- CKafka 面向的使用场景主要是需要高吞吐、高性能的大数据处理场景，对数据可靠性要求不十分苛刻，极端场景下可能会有少量的消息丢失；若需保障完全不丢失消息，且对性能要求不是非常高的场景，推荐使用 TDMQ。

CKafka 如何保证安全性?

CKafka 通过如下安全特性确保安全性：

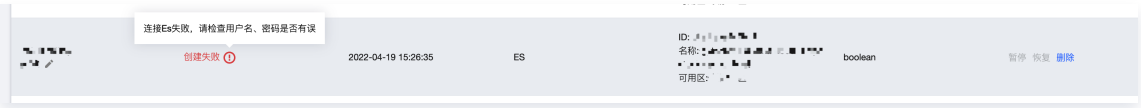
- 租户隔离：实例的网络访问在账户间默认隔离。
- 权限控制：CKafka 额外应用层上做了来源 IP 白名单的鉴权机制，支持 SASL 鉴权 。
- 安全防护：提供多纬度的安全防护、防 DDoS 攻击等服务。

连接器问题
任务相关问题

最近更新时间：2024-10-11 17:19:41

任务创建失败怎么处理？

转储的任务创建失败时，在控制台中会有一个提示。



一般情况下，能够通过这个提示检查：

1. 出现“连接XXX失败，请检查用户名、密码是否有误”，检查输入的用户名密码是否有误，然后重新创建任务输入正确的参数来解决创建失败的问题。
2. 出现“连接失败，请检查端口是否允许连接”，检查端口是否允许连接，可参见 Mongo Stream 数据变更记录分析 内的安全组开放端口。
3. 出现“数据库不存在这个表，表:xxx”，检查该数据库是否存在这个表。
4. 出现“createLink fail”，出现打通网络失败时，需要 联系我们 查看原因。

如果出现提示信息不清晰等无法知道具体创建失败的原因，可 联系我们 进行处理。

任务状态异常如何处理？

转储的任务发生异常时，在控制台中会有一个提示。
一般情况下，客户能通过异常信息的提示排查问题：

1. 出现“源 CKafka 实例不存在”或“目标 CKafka 实例不存在”时，查看是否当前实例被删除或者出现异常。
2. 出现“源 Topic 不存在”或“目标 Topic 不存在”时，查看是否为 CKafka 实例内的 Topic 是否被删除。
3. 出现“请检查账号、密码是否有误”，检查用户名密码是否更改过，需要重新去更改任务内的用户名密码。

如果出现提示信息不清晰等无法知道具体创建失败的原因，可 联系我们 进行处理。

任务并发度是什么？

每个任务（指非数据上报的任务）都会有一个并发度的概念，任务创建的时候默认的并发度是1。在任务运行过程中，系统会自动检测数据的堆积情况：

1. 当出现数据堆积时，系统会自动扩容并发度，加大数据的吞吐。
2. 当数据量低的时候，系统会缩容并发度，避免不必要的损耗。

在数据处理和数据转储的场景中：并发度最小是1，最大是 kafka topic 的分区数。
在数据接入场景中，并发度会根据上游引擎来调整，最小是1，最大暂无上限，视具体需求而定。

Mongo 接入的原理是什么？

Change Stream 是 MongoDB 记录数据变更的一种方式。当数据库中有任何数据发生变化，应用端都可以通过 change stream 机制得到变更信息。我们可以将其理解为在应用中执行的触发器。至于应用想得到什么数据，以什么形式得到数据，则可以通过聚合框架加以过滤和转换。
底层依赖的是mongo 官方的 MongoDB Kafka source connector 。connector会长期监听DB的变化信息，并将数据存储到kafka topic中。

Mongo的数据是分区有序的吗?

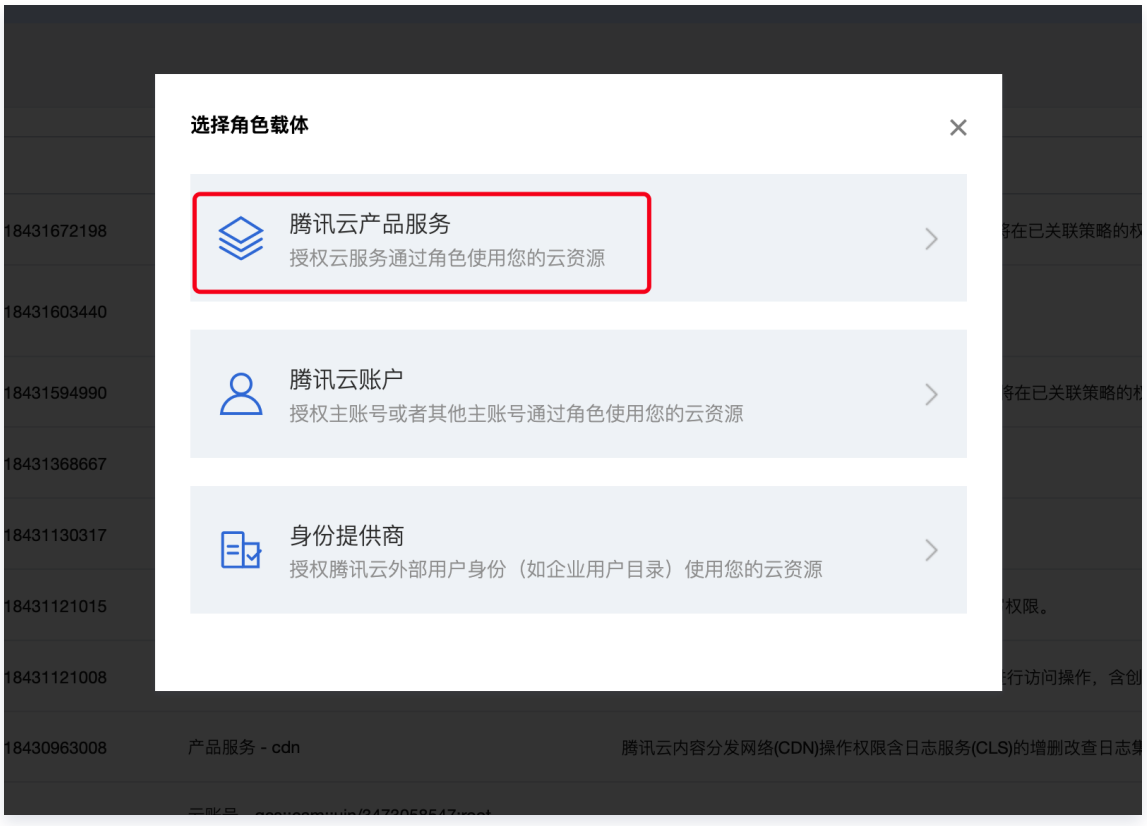
MongoDB 的数据默认情况下，是按照 MongoDB 的 object id 作为 key，写入到分区中的。所以在分区不变的情况下，是可以分区有序的。如果订阅过程中，扩容了分区，就会出现重新 hash 的情况，会出现消费时短暂的同一行记录的变更数据没有顺序。

Role 角色缺失怎么处理?

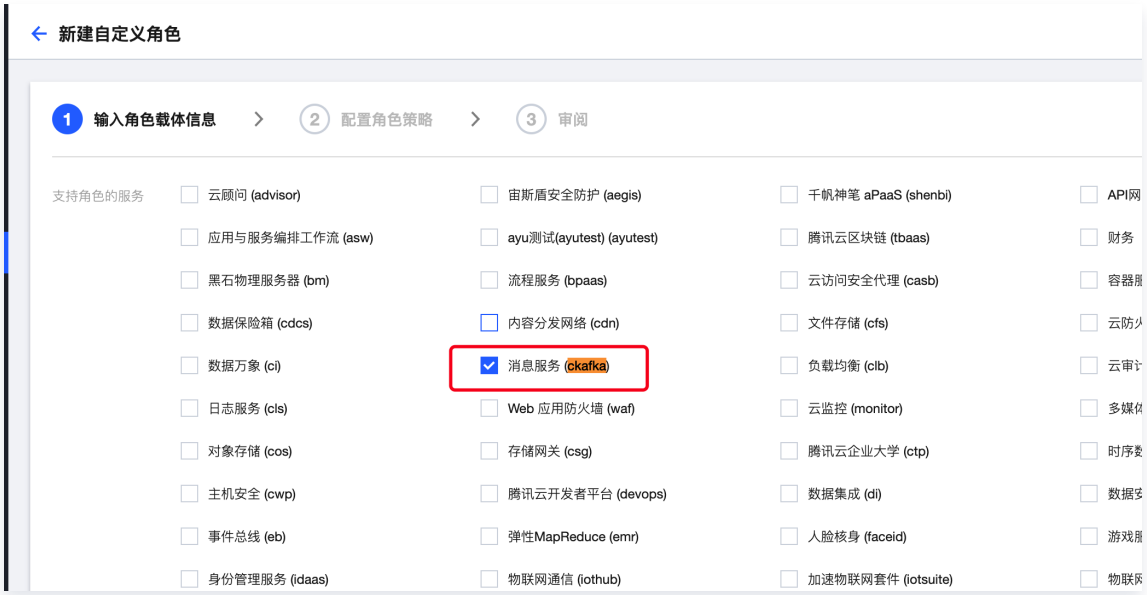
1. 在账号列表下找到访问管理。

2. 在左侧导航栏选择**角色**，单击**新建角色**。

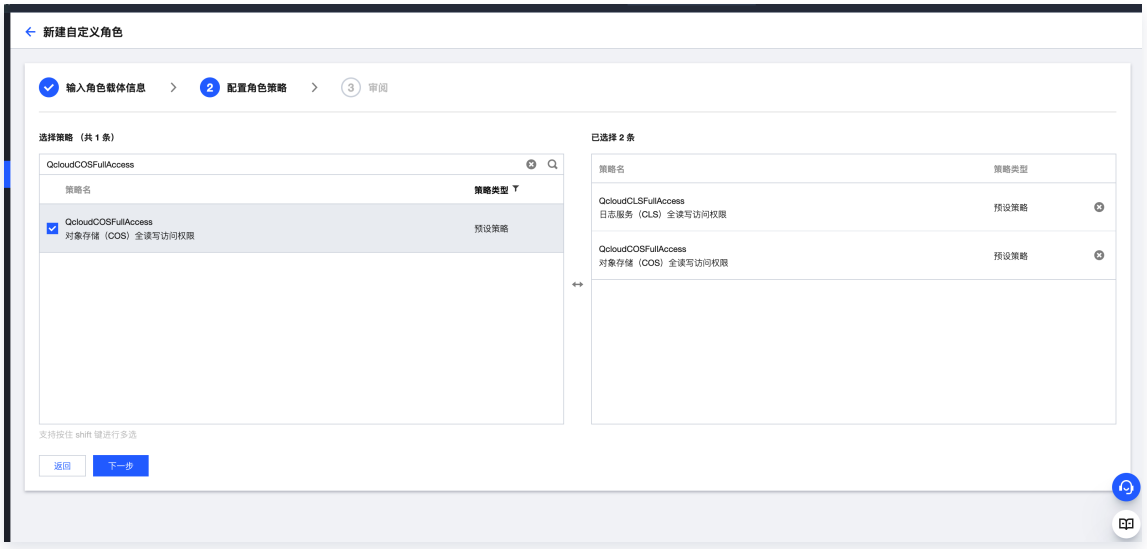
3. 角色载体选择**腾讯云产品服务**。



4. 在输入角色载体信息中找到**消息服务（ckafka）**。



5. 配置角色策略中，按照客户需要转储的服务，选择 QcloudCLSFullAccess、QcloudCOSFullAccess 等策略后单击**下一步**。



6. 在审阅页面输入角色名DIP_QcsRole，描述设置为当前角色为 CKafka 中连接器的服务角色，该角色将在已关联策略的权限范围内访问您的其他云服务资源。
7. 完成后使用 CKafka 转储数据。

数据上报相关问题

最近更新时间：2024-10-09 11:41:01

如何确认数据上报成功?

当请求的云 API 接口返回 MessageId 字段时，即代表数据已上报成功。参见云 API 调用返回 云 API 返回结果。

上报数据，出现错误怎么处理?

数据上报过程中，常见的云 API 错误返回结果如下：

```
{
    "Response": {
        "Error": {
            "Message": "TopicAuthorizationException: Not authorized to
access topics: [xxxxx]",
            "Code": "FailedOperation"
        },
        "RequestId": "xxxx9bef-52fd-4242-bdc0-8cf5e55xxxxxx"
    }
}
```

1. 根据报错信息可检查 Topic 是否设置了ACL 权限（目前 HTTP 数据上报暂不支持 ACL 设置）需要关闭前 Topic 所设置的 ACL。

```
{
    "Response": {
        "Error": {
            "Message": "task status suspended [datahub-lz7xxxxv]",
            "Code": "FailedOperation.ResourceTaskPaused"
        },
        "RequestId": "xxxxifef-942fd-4242-bdc0-z3f5e55xxxxxx"
    }
}
```

2. 根据错误消息内容看到当前任务状态处于暂停状态，恢复任务状态至运行中即可。

```
{
    "Response": {
        "Error": {
```

```
            "Message": "limit exceeded dataHubId [datahub-nyxxxxiy]",
            "Code": "RequestLimitExceeded"
        },
        "RequestId": "xxxxqsnrf-942fd-4242-nsdc0-z3f5e55xxxxxx"
    }
}
```

3. 每秒的请求数量到接口达上限。可根据业务提单调整当前的 QPS 限制。
4. 其他错误信息也可反馈完整的错误信息，可以 联系我们 处理。

数据上报配额是多少，如何提升配额?

数据上报是通过 HTTP 协议接入数据，HTTP 的接口的 QPS 限制默认是2000。当超时2000时，就会报 request limit 的限制。
如果需要更高的 QPS，可以 联系我们 处理，提升单个任务 ID 的 QPS 限制。

数据处理相关问题

最近更新时间：2024-10-11 15:39:01

数据处理过程中，因格式问题处理失败的数据会怎么处理?

数据处理过程中，因格式问题处理失败或不满足过滤器规则的数据会流向“失败处理”策略。
“失败处理”策略包括：

1. “丢弃”策略即丢弃处理失败的原始数据；
2. “保留”策略会保留处理失败的原始数据，并将原始数据和处理失败信息封装成新的消息体，投递到目标Topic；
3. “死信队列”策略会保留处理失败的原始数据，并将原始数据和处理失败信息封装成新的消息体，投递到死信队列。

数据处理过程如何知道处理的性能不足?

数据处理的流程分为3个阶段，分别是：

1. 消费数据源 Topic 的数据；
2. 对原始数据进行处理；
3. 将处理后数据投递到数据目标Topic。

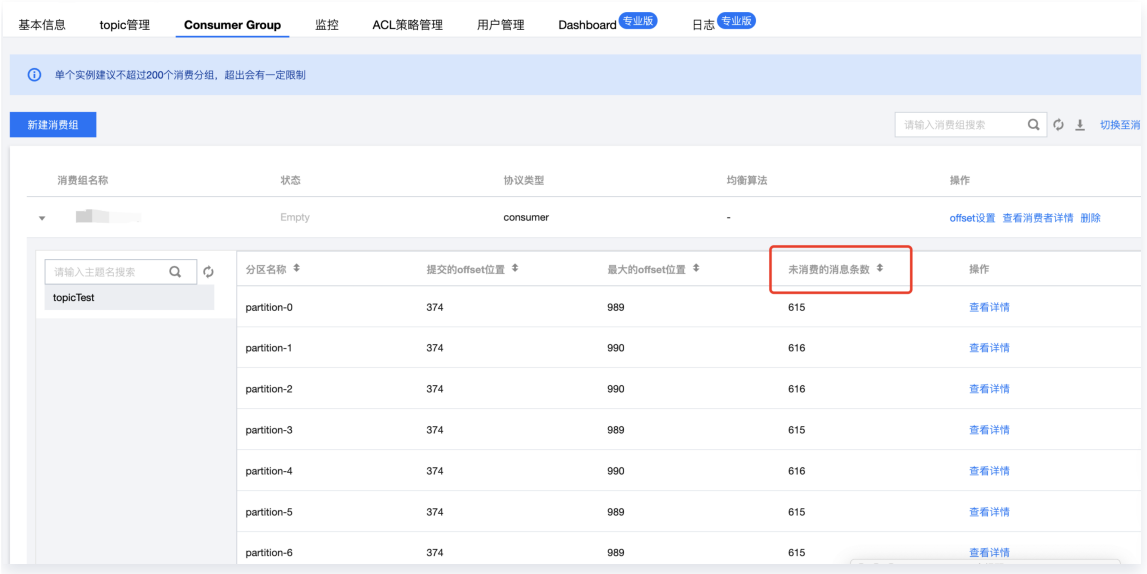
其中，我们可以通过消费者的消息积压程度和消费速度的监控来查看阶段1消费能力的性能，以及目标 Topic 的生产速度的监控来查看阶段3生产能力的性能。
一般我们可以通过提高数据源 Topic 的分区数，从而提高任务可扩容的并发度上限，以达到提升任务整体性能的效果。

数据转存相关问题

最近更新时间：2024-10-10 17:19:41

如何知道数据转储是否有堆积?

CKafka 数据转储，即 Kafka 的数据流出转储到其他源中，常见的例如 kafka to es，kafka to clickhouse 等等。
同步服务会消费 CKafka 实例的消息，因此会生成对应的消费分组，可在控制台的 ConsumerGroup 管理页面查看，一般消费分组命名为 datahub-task-xxx。 同步服务消费到消息后，会写入转储目标的服务中，然后提交写入条数对应的 offset 位置。
因此判断转储是否堆积，可通过查看该消费分组的未消费的消息条数是否在持续增加来了解堆积情况。



数据有堆积如何处理?

数据有堆积的情况一般分为两种：

- 一种是同步服务的消费能力受限，可提高任务并发度，后台同步服务会增加消费者的数量；或者适当增加 Topic 的分区数，提高消费者吞吐能力；如果实例的消费流量达到配额上限被限流，还需要升配实例的带宽规格。
- 另一种情况是在上面提升 Kafka 端消费能力后，堆积仍未有效改善。可能是写入流出源的速率受限，导致同步服务未能快速完成写入并提交 offset 的流程。例如 es 在大批写入达到瓶颈时可能会产生保护服务的锁拒绝外部写入甚至导致同步任务异常；或是 tdw 每秒写入条数达到上限被限制写入等场景。这个时候需要判断出流出源的写入瓶颈在哪里并调整提高流出源的写入速率。

数据订阅相关问题

最近更新时间：2024-10-11 17:19:41

MySQL 异常处理

配置和启动错误

以下几种情况会导致连接器启动失败，并在日志中报告错误或异常，然后停止运行：

- connector 的配置无效。
- connector 根据相关配置参数无法连接上 MySQL 服务器。
- connector 在重启后尝试从故障点恢复，但MySQL 的 binlog 被清理无相应历史记录。

当发生这些情况时，报错信息会提供错误的细节并可能给出一些处理方法，当您排查问题后可尝试重启 connector 服务。

MySQL 变得不可用

如果 MySQL 服务变得不可用时，connector 会停止工作，需要当 MySQL 可用时重启 connector 服务。如果您的 MySQL 集群使用了 GTIDs 协议，可立即重启 connector 服务，它会连接集群中的另一台服务器，并根据读取的最后一次提交事务处开始继续读取 binlog 。

Kafka Connect 异常终止

如果 Kafka Connect 在分布式模式下运行，并且 Kafka Connect 进程正常停止。在关闭该进程之前，Kafka Connect 将该进程的连接器任务迁移到该组中的另一个 Kafka Connect 进程。 新的 connector tasks 能够准确地从先前任务停止的位置开始继续处理。 在连接器任务正常停止并在新进程上重新启动时，处理会有短暂的延迟。

Kafka Connect 崩溃

当 kafka connect 发生崩溃时，进程立即停止并且来不及提交最近的偏移量，在分布式部署环境下， kafka connect 会重启新的进程，但新的进程无法获得崩溃进程最新的偏移量，因此可能会存在重复提交相同事件的情况。
但每一个更改事件消息都包括了 connector 的元数据信息，您可以使用这些信息来判别重复提交的事件。

Kafka 服务不可用

当 kafka 服务变的不可用时，Debezium MySQL connector 会暂停直到它和 kafka 服务重新建立连接。

MySQL 清理 binlog 文件

如果 Debezium MySQL connector 停止时间过长，MySQL 服务器可能会清理 binlog 从而导致 connector 读取的最新位置被清理。当 connector 重启时如果发现原始读取位置被清理，会尝试重新初始化 snapshot，如果 snapshot 被禁止，则 connector 会异常终止。

PostgreSQL 异常处理

配置和启动错误

以下几种情况会导致连接器启动失败，并在日志中报告错误或异常，然后停止运行：

- connector 的配置无效。
- connector 根据相关配置参数无法连接上 PostgreSQL 服务器。
- connector 在重启时尝试从故障发生时的记录偏移位重新继续读取，但 PostgreSQL 已经清理了相关的记录。

当发生这些情况时，报错信息会提供错误的细节并可能给出一些处理方法，当您排查问题后可尝试重启 connector 服务。

PostgreSQL 变得不可用

如果 PostgreSQL 服务变得不可用时，connector 会停止工作，需要当 PostgreSQL 可用时重启 connector 服务。

集群故障

发布版本 12 后，PostgreSQL 只允许在主服务器上设置逻辑复制槽。 这意味着您只能将 Debezium PostgreSQL connector 指向数据库集群的主服务器。 此外，复制槽本身不会传播到副本。 如果主服务器出现故障，则必须选举新的主服务器。
新的主服务器必须安装 logical decoding plug-in ，配置为供插件使用的复制槽，运行要捕获更改事件的数据库。 只有这样，您才能将连接器配置连接新服务器并重新启动 connector。
发生故障转移时有一些重要的警告，您应该暂停 Debezium 服务，并在验证有一个完整的复制槽并且没有丢失数据后重启服务。 故障转移后：

- 在允许应用程序写入新的主节点之前，必须有一个重新创建 Debezium 复制槽的进程，否则应用程序可能会错过更改事件。
- 您可能需要验证 Debezium 是否能够在旧主节点终止之前读取插槽中的所有更改。

恢复和验证是否有更改事件丢失的一种可靠方法是将故障主节点的备份恢复到故障前的位置。 虽然这在执行上可能很困难，但它允许您检查复制槽是否有任何未消费的更改。

Kafka Connect 异常终止

如果 Kafka Connect 在分布式模式下运行，并且 Kafka Connect 进程正常停止。在关闭该进程之前，Kafka Connect 将该进程的连接器任务迁移到该组中的另一个 Kafka Connect 进程。 新的 connector tasks 能够准确地从先前任务停止的位置开始继续处理。 在连接器任务正常停止并在新进程上重新启动时，处理会有短暂的延迟。

Kafka Connect 崩溃

如果 Kafka 连接器进程意外停止，它正在运行的任何连接器任务都会终止，并且不会记录它们最近处理的偏移量。当 Kafka Connect 在分布式模式下运行时，Kafka Connect 会在其他进程上重新启动这些连接器任务。但是

PostgreSQL 连接器从早期进程记录的最后一个偏移量恢复。这意味着新的替换任务可能会生成一些在崩溃之前处理过的相同更改事件。重复事件的数量取决于偏移刷新周期和崩溃前的数据量变化。
在每个更改事件记录中，Debezium 连接器记录了事件起源相关的元数据信息，包括事件发生时 PostgreSQL 服务器时间、服务器事务的 ID 。消费者可以跟踪此信息，尤其是 LSN，以确定事件是否重复。

Kafka 服务不可用

当 kafka 服务变的不可用时，Debezium PostgreSQL connector 会暂停并重试连接，直到它和 kafka 服务重新建立连接。

Connector 停止服务了一段时间

如果连接器正常停止，则可以继续使用数据库。 任何更改都会记录在 PostgreSQL WAL 中。 当连接器重新启动时，它会从中断的地方继续提交更改。也就是说，它会为连接器停止时所做的所有数据库更改生成更改事件记录。
合理配置的 Kafka 集群能够以超大吞吐量处理数据。 Kafka Connect 是根据 Kafka 最佳实践编写的，如果有足够的资源，Kafka Connect 连接器也可以处理非常大量的数据库更改事件。 因此，在停止一段时间后，当 Debezium 连接器重新启动时，它很可能会赶上停止时数据库发生的更改。