消息队列 CMQ

快速入门

腾讯云

文档目录

快速入门
入门指南

最近更新时间：2020-07-31 17:39:05

腾讯云消息队列（Cloud Message Queue，CMQ）是分布式消息队列服务，能够为分布式部署的不同应用之间或者一个应用的不同组件之间，提供基于消息的可靠的异步通信机制，消息被存储在高可靠、高可用的 CMQ 队列中，多进程可以同时读写、互不干扰。
CMQ 提供了四种 SDK，本文以 Python 为例进行说明。

Python SDK 简介

为了方便使用，CMQ 将用户操作、队列操作、主题操作等抽象成了几个类：

- Account：封装账户 SecretId、SecretKey，用户可以创建删除队列、主题及订阅，并查看这些对象。
- queue：收发消息，查看设置队列属性。
- topic：发布消息，查看设置主题属性，查看订阅者。
- cmq_client：可以设置一些客户端的与服务器端连接属性，如设置是否写日志、连接超时时间、是否长连接等。

> ⚠ **注意**
> 所有的类均为非线程安全的，如果需要多线程使用，最好每个线程实例化自己的对象。

下载 SDK >>

队列模型

这里的队列与数据结构中定义的 Queue 有一定区别。数据结构中的队列严格按照 FIFO 操作，这里的分布式队列不会有严格的 FIFO（后续会推出专属的 FIFO 产品）。这里的队列相当于一个高性能、高容量、高可靠的容器，可以生产消息投递进去，也可以把消息取出来消费。队列在初始化的时候有自己的属性设置。属性以及含义如下：

属性	描述
maxMsgHeapNum	最大堆积消息数。队列中可以存储的消息条数，该属性表示了队列的存储和堆积能力。
pollingWaitSeconds	消息接收长轮询等待时间。取值范围0 – 30秒，该时间设置表示的是消费消息时默认等待接收消息的时间。 例如设置为10，那么在消费消息时，如果没有消息， 默认会等待10s返回；如果有消息则会立即返回。 也可以在接收消息的时候设置自定义的等待时间，不使用队列的属性值。

visibilityTimeout	消息可见性超时。 消息被消费者获取到，会有一个不可见时间，即在这个时间之内别的消费者无法获取这条消息。取值范围1 - 43200秒（即12小时内），默认值为30。
maxMsgSize	消息最大长度。取值范围1024 - 1048576Byte（即1K - 1024K），默认值为65536。
msgRetentionSeconds	消息生命周期，即消息在队列中的保存时间，取值范围60 - 1296000秒（1min - 15天），默认值为345600（4天）。
createTime	队列的创建时间。返回 Unix 时间戳，精确到秒。
lastModifyTime	最后一次修改队列属性的时间。返回 Unix 时间戳，精确到秒。
activeMsgNum	在队列中处于 Active 状态（不处于被消费状态）的消息总数，为近似值。
inactiveMsgNum	在队列中处于 Inactive 状态（正处于被消费状态）的消息总数，为近似值。
rewindSeconds	回溯队列的消息回溯时间最大值，取值范围0 - 43200秒，0表示不开启消息回溯。
rewindmsgNum	已调用 DelMsg 接口删除但还在回溯保留时间内的消息数量。
minMsgTime	消息最小未消费时间，单位为秒。
delayMsgNum	延时消息数量。

查看队列模型快速入门>>

主题模型

主题模型类似设计模式中的发布订阅模式，Topic 相当于发布消息的单位，Topic 下面的订阅者相当于观察者模式。Topic 会把发布的消息主动推送给订阅者：

属性	描述
msgCount	当前该主题中堆积的消息数目（消息堆积数）。
maxMsgSize	消息最大长度。取值范围1024 - 1048576Byte（即1 - 1024KB），默认值为65536。

msgRetentionSeconds	消息在主题中最长存活时间，从发送到该主题开始经过此参数指定的时间后，不论消息是否被成功推送给用户都将被删除，该参数单位为秒。固定为一天（86400秒），该属性不能修改。
createTime	主题的创建时间。返回 Unix 时间戳，精确到秒。
lastModifyTime	最后一次修改主题属性的时间。返回 Unix 时间戳，精确到秒。
filterType	描述用户创建订阅时选择的过滤策略： • filterType = 1 表示用户使用 filterTag 标签过滤 • filterType = 2 表示用户使用 bindingKey 过滤

查看主题模型快速入门 >>

队列模型快速入门

最近更新时间：2021-08-04 15:20:19

1. 创建队列

```
endpoint='' //CMQ 的域名
secretId ='' // 用户的 ID 和 key
secretKey = ''
      account = Account(endpoint,secretId,secretKey)
      queueName = 'QueueForTest'
      queue=account.get_queue(queueName)
      queue_meta = QueueMeta()
      queue_meta.queueName = queueName
      queue_meta.visibilityTimeout = 10
      queue_meta.maxMsgSize = 65536
      queue_meta.pollingWaitSeconds = 10
      try:
          queue.create(queue_meta)
      except CMQExceptionBase,e:
          print e
```

创建队列成功后，您可以从控制台查看已创建的队列信息。

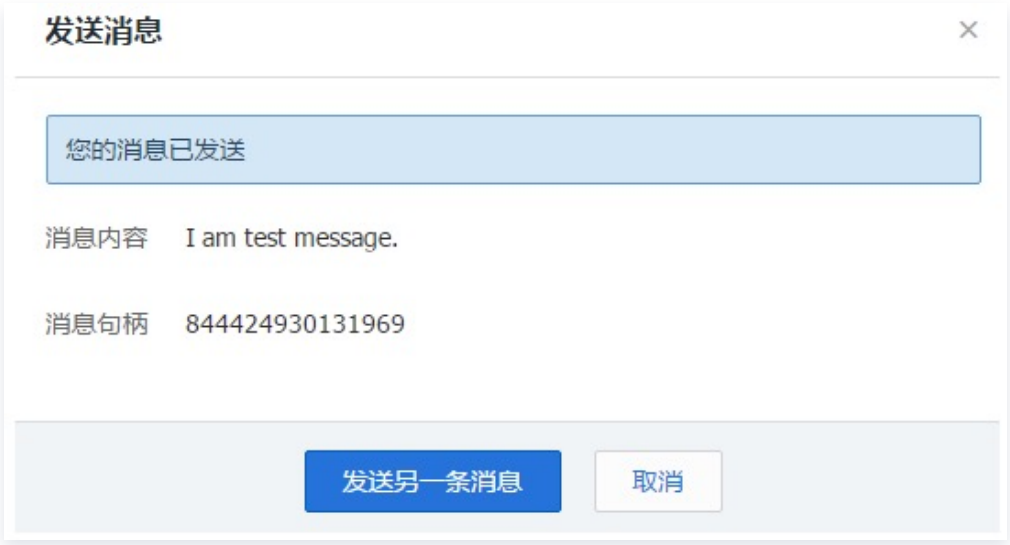
2. 生产消息

用户只要取得队列对象，就可以调用队列的发送消息接口，将消息发送到队列中。发送消息有两种方式：单独发送和批量发送。

- **生产消息：**

```
msg_body = "I am test message."
msg = Message(msg_body)
re_msg = my_queue.send_message(msg)
```

您可以直接从控制台查看消息属性。



- **批量生产消息：**

```
msg_count=3
messages=[]
for i in range(msg_count):
  msg_body = "I am test message %s." % i
  msg = Message(msg_body)
  messages.append(msg)
re_msg_list = my_queue.batch_send_message(messages)
```

3. 消费消息

pollingWaitSeconds 为消费消息中的默认参数，该参数表示消费消息愿意等待的时间，如果不填该参数，则会默认使用队列中的属性值。

- **消费消息：**

```
wait_seconds=3
recv_msg = my_queue.receive_message(wait_seconds)
```

- **批量消费消息：**

```
wait_seconds = 3
num_of_msg = 3
recv_msg_list = my_queue.batch_receive_message(num_of_msg,
wait_seconds)
```

注意事项

- **设置恰当的 pollingWaitSeconds**
  这里的 pollingWaitSeconds，可以使用自定义的值，也可以使用队列默认值。如果将值设置为0，表示不等待消息。但设置为0有可能会造成空消息的现象（但实际队列中存在消息），这是由于当大量的消费者同时消费消息时有可能需要排队等待队列服务，如果设置了等待时间为0，可能根本没有等到队列服务就已经超时，返回 no message 异常了，因此建议消息的等待时间不要设置为0。
- **队列消息数 < 批量消费数，不会阻塞本次消费操作**
  批量消费消息的时候，需要填写本次接收消息的个数，如果队列中的消息数小于本次需要消费的消息数，不会阻塞本次操作。
- **队列属性中设置：不可见时间 > 消息堆积周期，实现每个消息均会消费且只消费一次**
  当不可见时间 > 消息堆积周期时，消息被消费之后会永远不可见，直到超过堆积周期被队列删除，这样看着确实消息只被消费了一次，且不会被再消费了。
  但是生产和消费的过程中有可能存在重复生产和消费失败等现象，只通过修改队列属性无法保证队列只消费一次，这里需要业务方来实现生产消费的去重和容错。详请参见 消息去重 。

> **说明**
> 不可见时间计时方法为：在消费者消费了消息后，不可见时间才开始计时，则不可见时间为消息被消费之后直到现在的时间。

4. 消息回溯

这里结合一个场景来描述消息回溯的使用：
假设有 A/B 两个业务，正常的生产消费场景，A 生产消息投递到队列中，B 从队列中消费消息，A/B 这时已经实现互相解耦，双方互不关心。A 只需要生产消息投递即可，B 从队列中拿到消息，然后将消息从队列中删除，接着在本地消费消息。
如果出现了异常，例如 B 业务虽然进行了消费，但是在一段时间内消费情况都出现了异常。这时，已经删除的消息已经被删除，无法重新消费，会对业务造成影响，且需要暂停 B 业务，等开发运维人员修复 Bug 之后才能重新上线 B 业务。而且运维人员也无法实时监控 B 业务的情况，等到发现异常场景，可能已经过去一段时间。
为了防止这种情况出现，A 业务需要关心 B 业务的处理情况，需要对生产消息进行备份，确保B 业务正常才能删除备份，保证现网正常。
这种情况下，您可以使用 **消息回溯** 的功能，开发人员对 B 业务进行修复，然后根据 B 业务消费正常的最近一个时间点，将消息回溯到该时间点。这时候，B 业务获取到的消息就会从指定的时间点开始，A 业务完全不用关心 B 业务的异常情况。这里 B 需要注意要对消费进行幂等处理。
了解消息回溯功能>>

开启消息回溯功能

```
endpoint='' //CMQ 的域名
secretId ='' // 用户的 ID 和 key
secretKey = ''
account = Account(endpoint,secretId,secretKey)
```

```
queueName = 'QueueTest'
my_queue = account.get_queue(queueName)
queue_meta = QueueMeta()
queue_meta.rewindSeconds = 43200 //消息允许回溯的时间，单位为秒
my_queue.create(queue_meta)
```

使用消息回溯

```
my_queue.rewindQueue(1488718862) //本次消息回溯的时间点，为 Unix 时间戳
```

5. 延时消息

延时消息是指生产消息的时候，可以指定一个飞行时间，即消息投递到队列中的花费时间。只有过了该消费时间，消息才会被消费者消费。
很多业务都有失败的场景出现，而失败之后需要进行重试，大多数业务不会立刻进行消费重试，而会隔一段时间再重试消费，这时就可以使用延时消息的功能。
示例：

```
message_body='i am test'
msg = Message(message_body)
my_queue.send_message(msg)
//这时候发现消息消费失败。可以重新投递消息并设置消息的飞行时间。
my_queue.send_message(msg,600) //这里设置飞行时间为10分钟。
//可以通过消息属性查看当前队列中的延时消息数量
queue_meta = my_queue.get_attributes()
print queue_meta.delayMsgNum
```

主题模型快速入门

最近更新时间：2021-08-04 15:17:52

主题发布消息有一个前提，即需要有订阅者订阅主题，如果没有订阅者存在，那么主题中的消息不会被投递，此时发布消息这一操作就失去了意义。

1. 创建主题

```
endpoint='' //CMQ 的域名
secretId ='' // 用户的 ID 和 key
secretKey = ''
account = Account(endpoint,secretId,secretKey)
topicName = 'TopicTest8B'
my_topic = account.get_topic(topicName)
topic_meta = TopicMeta()
my_topic.create(topic_meta)
```

您可以从控制台查看已创建的主题。其中 QPS = 5000，表示调用同一个接口的频率上限，默认为5000次/s。如果您需要提高 QPS 上限，可以通过 提交工单 反馈给我们。

主题名	TopicTest8B
主题名ID	topic-088nhan8
消息生命周期	24 h
消息最大长度	64 KB
消息堆积	开启
堆积消息数	1 条
QPS	5,000
创建时间	2016-12-01 22:11:35
修改时间	2016-12-01 22:11:35

2. 发布消息

您可以通过 SDK 或控制台两种方式发布消息。

使用 SDK 发送消息

```
message = Message()
message.msgBody = "this is a test message"
my_topic.publish_message(message)
```

使用控制台发布消息

Topic 目前支持消息过滤，即消息标签、消息类型，用来区分某个 CMQ 的 Topic 下的消息分类。MQ 允许消费者按照标签对消息进行过滤，确保消费者最终只消费到他关心的消息类型。该功能默认不开启，未开启时，所有消息向所有订阅者发送；增加标签后，订阅者将仅能收到带该标签的信息。消息过滤标签描述了该订阅中消息过滤的标签（标签一致的消息才会被推送）。单个标签不超过16个字符，单个 Message 可最多添加5个标签。
Topic 目前支持标签过滤和 routingKey 过滤两种方式。过滤规则如下图所示：

批量发布消息：

```
vmsg = []
for i in range(6):
message = Message()
message.msgBody = "this is a test message"
vmsg.append(message)

my_topic.batch_publish_message(vmsg)
```

3. 消息处理

主题发布消息之后，会自动将消息推送给订阅，当推送失败时，有两种重试策略：

- **退避重试**：重试3次，间隔时间为10 – 20s之间的一个随机值，超过3次后，该条消息对于该订阅者丢弃，不会再重试。
- **衰退指数重试**：重试176次，总计重试时间为1天，间隔时间依次为：2^0 , 2^1, …, 512, 512, …, 512秒。默认为衰退指数重试策略。

使用队列处理消息

订阅者可以填写一个 Queue，使用队列来接收发布的消息。

```
subscription_name = "subsc-test"
my_sub = my_account.get_subscription(topic_name, subscription_name)
subscription_meta = SubscriptionMeta()
# 填写订阅名称，这里填写队列名称
subscription_meta.Endpoint = "queue name  "
subscription_meta.Protocal = "queue"
my_sub.create(subscription_meta)
```

使用其他方式处理消息

订阅者也可以不与 Queue 结合，自己来处理消息。详情请参考 投递消息 。

4. 路由匹配

Binding key 、Routing key 是组合使用的，兼容 rabbitmq topic 匹配模式。发消息时配的 Routing key 是客户端发消息带的， 必须为字符串，不能有匹配符。创建订阅关系时配的 Binding key 是 topic 和订阅者的绑定关系。

使用限制：

- Binding key 的数量不超过5个。单个 binding key 的长度 ⩽ 64字节，用于表示发送消息的路由路径，最多含有15个“.”，即最多16个词组。
- Routing key 的数量由1个字符串组成。单个 Routing key 的长度 ⩽ 64字节，用于表示发送消息的路由路径，最多含有15个“.”，即最多16个词组。

通配符说明：

- *（星号）：可以替代一个单词（一串连续的字母串）。
- #（井号）：可以匹配一个或多个字符。
- rabbitmq的特例：routing_key为空字符串时，不匹配 * ，但可以匹配 #。

示例：

- 订阅者是『1.*.0』，此时消息为『1.任意字符.0』，则订阅者都能收到消息。
- 订阅者是『1.#.0』，此时消息为『1.2.3.4.4.2.2.0』，则订阅者都能收到消息（消息中间元素随意）。

使用路由匹配功能

```
    endpoint='' //CMQ 的域名
    secretId ='' // 用户的 ID 和 key
    secretKey = ''
    account = Account(endpoint,secretId,secretKey)
    topicName = 'TopicTest'
    my_topic = account.get_topic(topicName)
    topic_meta = TopicMeta()
    topic_meta.filterType =2 //表示消息投递给订阅的时候采用路由匹配
    //若 filterType =1 则表示使用标签过滤
    my_topic.create(topic_meta)

    subscription_name = "subsc-test"
    my_sub = my_account.get_subscription(topic_name, subscription_name)
    subscription_meta = SubscriptionMeta()
    //填写订阅名称，这里填写队列名称
    subscription_meta.Endpoint = "queue name  "
    subscription_meta.Protocal = "queue"
    subscription_meta.bindingKey=[1.*.0]  //若消息的标签为[1.任意字符.0]，则
该订阅者都能收到该标签
    my_sub.create(subscription_meta)
```

发布消息

```
    message = Message()
    message.msgBody = "this is a test message"
    message.routingKey = '1.test.0' //该消息会被投递到 my_sub 订阅的地址
    my_topic.publish_message(message)
```