重要的生产者参数

在 KafkaProducer 中，除了第3节提及的3个默认的客户端参数，大部分的参数都有合理的默认值，一般不需要修改它们。不过了解这些参数可以让我们更合理地使用生产者客户端，其中还有一些重要的参数涉及程序的可用性和性能，如果能够熟练掌握它们，也可以让我们在编写相关的程序时能够更好地进行性能调优与故障排查。下面挑选一些重要的参数进行讲解。

1. acks

这个参数用来指定分区中必须要有多少个副本收到这条消息，之后生产者才会认为这条消息是成功写入的。acks 是生产者客户端中一个非常重要的参数，它涉及消息的可靠性和吞吐量之间的权衡。acks 参数有3种类型的值（都是字符串类型）。

- acks = 1。默认值即为1。生产者发送消息之后，只要分区的 leader 副本成功写入消息，那么它就会收到来自服务端的成功响应。如果消息无法写入 leader 副本，比如在 leader 副本崩溃、重新选举新的 leader 副本的过程中，那么生产者就会收到一个错误的响应，为了避免消息丢失，生产者可以选择重发消息。如果消息写入 leader 副本并返回成功响应给生产者，且在被其他 follower 副本拉取之前 leader 副本崩溃，那么此时消息还是会丢失，因为新选举的 leader 副本中并没有这条对应的消息。acks 设置为1，是消息可靠性和吞吐量之间的折中方案。
- acks = 0。生产者发送消息之后不需要等待任何服务端的响应。如果在消息从发送到写入 Kafka 的过程中出现某些异常，导致 Kafka 并没有收到这条消息，那么生产者也无从得知，消息也就丢失了。在其他配置环境相同的情况下，acks 设置为0可以达到最大的吞吐量。
- acks = -1 或 acks = all。生产者在消息发送之后，需要等待

ISR 中的所有副本都成功写入消息之后才能够收到来自服务端的成功响应。在其他配置环境相同的情况下，acks 设置为 -1（all） 可以达到最强的可靠性。但这并不意味着消息就一定可靠，因为ISR中可能只有 leader 副本，这样就退化成了 acks=1 的情况。要获得更高的消息可靠性需要配合 min.insync.replicas 等参数的联动，消息可靠性分析的具体内容可以参考《图解Kafka之核心原理》。

注意 acks 参数配置的值是一个字符串类型，而不是整数类型。举个例子，将 acks 参数设置为0，需要采用下面这两种形式：

```
properties.put("acks", "0");
# 或者
properties.put(ProducerConfig.ACKS_CONFIG, "0");
```

而不能配置成下面这种形式：

```
properties.put("acks", 0);
# 或者
properties.put(ProducerConfig.ACKS_CONFIG, 0);
```

这样会报出如下的异常：

```
org.apache.kafka.common.config.ConfigException:
Invalid value 0 for configuration acks: Expected
value to be a string, but it was a
java.lang.Integer.
```

2. max.request.size

这个参数用来限制生产者客户端能发送的消息的最大值，默认值为1048576B，即1MB。一般情况下，这个默认值就可以满足大多数的应用场景了。

笔者并不建议读者盲目地增大这个参数的配置值，尤其是在对 Kafka 整体脉络没有足够把控的时候。因为这个参数还涉及一些其他参数的联动，比如 broker 端的 message.max.bytes 参数，如果配置错误可能会引起一些不必要的异常。比如将 broker 端的 message.max.bytes 参数配置为10，而 max.request.size 参数配置为20，那么当我们发送一条大小为15B的消息时，生产者客户端就会报出如下的异常：

```
org.apache.kafka.common.errors.RecordTooLargeException: The request included a message larger than the max message size the server will accept.
```

3. retries和retry.backoff.ms

retries 参数用来配置生产者重试的次数，默认值为0，即在发生异常的时候不进行任何重试动作。消息在从生产者发出到成功写入服务器之前可能发生一些临时性的异常，比如网络抖动、leader 副本的选举等，这种异常往往是可以自行恢复的，生产者可以通过配置 retries 大于0的值，以此通过内部重试来恢复而不是一味地将异常抛给生产者的应用程序。如果重试达到设定的次数，那么生产者就会放弃重试并返回异常。不过并不是所有的异常都是可以通过重试来解决的，比如消息太大，超过 max.request.size 参数配置的值时，这种方式就不可行了。

重试还和另一个参数 retry.backoff.ms 有关，这个参数的默认值为100，它用来设定两次重试之间的时间间隔，避免无效的频繁重试。在配置 retries 和 retry.backoff.ms 之前，最好先估算一下可能的异常恢复时间，这样可以设定总的重试时间大于这个异常恢复时间，以此来避免生产者过早地放弃重试。

Kafka 可以保证同一个分区中的消息是有序的。如果生产者按照一定的顺序发送消息，那么这些消息也会顺序地写入分区，进而消费者也可以按照同样的顺序消费它们。

对于某些应用来说，顺序性非常重要，比如 MySQL 的 binlog 传输，如果出现错误就会造成非常严重的后果。如果将acks参数配置为非零值，并且 max.in.flight.requests.per.connection 参数配置为大于1的值，那么就会出现错序的现象：如果第一批次消息写入失败，而第二批次消息写入成功，那么生产者会重试发送第一批次的消息，此时如果第一批次的消息写入成功，那么这两个批次的消息就出现了错序。一般而言，在需要保证消息顺序的场合建议把参数 max.in.flight.requests.per.connection 配置为1，而不是把 acks 配置为0，不过这样也会影响整体的吞吐。

4. compression.type

这个参数用来指定消息的压缩方式，默认值为“none”，即默认情况下，消息不会被压缩。该参数还可以配置为“gzip”“snappy”和“lz4”。对消息进行压缩可以极大地减少网络传输量、降低网络I/O，从而提高整体的性能。消息压缩是一种使用时间换空间的优化方式，如果对时延有一定的要求，则不推荐对消息进行压缩。

5. connections.max.idle.ms

这个参数用来指定在多久之后关闭闲置的连接，默认值是540000（ms），即9分钟。

6. linger.ms

这个参数用来指定生产者发送 ProducerBatch 之前等待更多消息（ProducerRecord）加入 ProducerBatch 的时间，默认值为0。生产者客户端会在 ProducerBatch 被填满或等待时间超过 linger.ms 值时发送出去。增大这个参数的值会增加消息的延迟，但是同时能提升一定的吞吐量。这个 linger.ms 参数与 TCP 协议中的 Nagle 算法有异曲同工之妙。

7. receive.buffer.bytes

这个参数用来设置 Socket 接收消息缓冲区（SO_RECBUF）的大小，默认值为32768（B），即32KB。如果设置为-1，则使用操作系统的默认值。如果 Producer 与 Kafka 处于不同的机房，则可以适地调大这个参数值。

8. send.buffer.bytes

这个参数用来设置 Socket 发送消息缓冲区（SO_SNDBUF）的大小，默认值为131072（B），即128KB。与 receive.buffer.bytes 参数一样，如果设置为-1，则使用操作系统的默认值。

9. request.timeout.ms

这个参数用来配置 Producer 等待请求响应的最长时间，默认值为30000（ms）。请求超时之后可以选择进行重试。注意这个参数需要比 broker 端参数 replica.lag.time.max.ms 的值要大，这样可以减少因客户端重试而引起的消息重复的概率。

还有一些生产者客户端的参数在本节中没有提及，这些参数同样非常重要，它们需要单独的章节或场景来描述。部分参数在前面的章节中已经提及，比如 bootstrap.servers，还有部分参数会在后面的章节中提及，比如 transactional.id。表中罗列了一份详细的参数列表以供读者参阅。

参 数 名 称	
bootstrap.servers	“”
key.serializer	“”
value.serializer	“”

buffer.memory	33554432（32MB）
batch.size	16384（16KB）
client.id	“”
max.block.ms	60000
partitioner.class	org.apache.kafka.clients
enable.idempotence	false
interceptor.classes	“”
max.in.flight.requests.per.connection	5
metadata.max.age.ms	300000（5分钟）
transactional.id	null

到目前为止主要讲述了生产者客户端的具体用法及其整体架构，主要内容包括配置参数的详解、消息的发送方式、序列化器、分区器、拦截器等。在实际应用中，一套封装良好的且灵活易用的客户端可以避免开发人员重复劳动，也提高了开发效率，还可以提高程序的健壮性和可靠性，而 Kafka 的客户端正好包含了这些特质。对于 KafkaProducer 而言，它是线程安全的，我们可以在多线程的环境中复用它，而对于下面要讲解的消费者客户端 KafkaConsumer 而言，它是非线程安全的，因为它具备了状态，具体怎么使用我们不妨继续来了解下面的内容。