目录

Property	Default	Description

broker.id		每个 broker 都可以用一个唯一的非负整数 id 进行标识；这个 id 可以作为 broker 的“名字”，并且它的存在使得 broker 无须混淆 consumers 就可以迁移到不同的 host/port 上。你可以选择任意你喜欢的数字作为 id，只要 id 是唯一的即可。
log.dirs	/tmp/kafka-logs	kafka 存放数据的路径。这个路径并不是唯一的，可以是多个，路径之间只需要使用逗号分隔即可；每当创建新 partition 时，都会选择在包含最少 partitions 的路径下进行。
port	9092	server 接受客户端连接的端口
zookeeper.connect	localhost:2181	ZooKeeper 连接字符串的格式为：hostname:port，此处 hostname 和 port 分别是 ZooKeeper 集群中某个节点的 host 和 port；为了当某个 host 宕掉之后你能通过其他 ZooKeeper 节点进行连接，你可以按照一下方式制定多个 hosts：hostname1:port1, hostname2:port2, hostname3:port3. ZooKeeper 允许你增加一个“chroot”路径，将集群中所有 kafka 数据存放在特定的路径下。当多个 Kafka 集群或者其他应用使用相同

		ZooKeeper 集群时，可以使用这个方式设置数据存放路径。这种方式的实现可以通过这样设置连接字符串格式，如下所示： hostname1：port1，hostname2：port2，hostname3：port3/chroot/path 这样设置就将所有 kafka 集群数据存放在 /chroot/path 路径下。注意，在你启动 broker 之前，你必须创建这个路径，并且 consumers 必须使用相同的连接格式。
message.max.bytes	1000000	server 可以接收的消息最大尺寸。重要的是，consumer 和 producer 有关这个属性的设置必须同步，否则 producer 发布的消息对 consumer 来说太大。
num.network.threads	3	server 用来处理网络请求的网络线程数目；一般你不需要更改这个属性。
num.io.threads	8	server 用来处理请求的 I/O 线程的数目；这个线程数目至少要等于硬盘的个数。
background.threads	4	用于后台处理的线程数目，例如文件删除；你不需要更改这个属性。
queued.max.requests	500	在网络线程停止读取新请求之前，可以排队等待 I/O 线程处理的最大请求个数。

host.name	null	broker 的 hostname；如果 hostname 已经设置的话，broker 将只会绑定到这个地址上；如果没有设置，它将绑定到所有接口，并发布一份到 ZK
advertised.host.name	null	如果设置，则就作为 broker 的 hostname 发往 producer、consumers 以及其他 brokers
advertised.port	null	此端口将给与 producers、consumers、以及其他 brokers，它会在建立连接时用到； 它仅在实际端口和 server 需要绑定的端口不一样时才需要设置。
socket.send.buffer.bytes	100 * 1024	SO_SNDBUFF 缓存大小，server 进行 socket 连接所用
socket.receive.buffer.bytes	100 * 1024	SO_RCVBUFF 缓存大小，server 进行 socket 连接时所用
socket.request.max.bytes	100 * 1024 * 1024	server 允许的最大请求尺寸； 这将避免 server 溢出，它应该小于 Java heap size
num.partitions	1	如果创建 topic 时没有给出划分 partitions 个数，这个数字将是 topic 下 partitions 数目的默认数值。

log.segment.bytes	1014*1024*1024	topic partition 的日志存放在某个目录下诸多文件中，这些文件将 partition 的日志切分成一段一段的；这个属性就是每个文件的最大尺寸；当尺寸达到这个数值时，就会创建新文件。此设置可以由每个 topic 基础设置时进行覆盖。 查看 the per-topic configuration section
log.roll.hours	24 * 7	即使文件没有到达 log.segment.bytes，只要文件创建时间到达此属性，就会创建新文件。这个设置也可以有 topic 层面的设置进行覆盖； 查看 the per-topic configuration section
log.cleanup.policy	delete	
log.retention.minutes 和 log.retention.hours	7 days	每个日志文件删除之前保存的时间。默认数据保存时间对所有 topic 都一样。 log.retention.minutes 和 log.retention.bytes 都是用来设置删除日志文件的，无论哪个属性已经溢出。 这个属性设置可以在 topic 基本设置时进行覆盖。 查看 the per-topic configuration section
log.retention.bytes	-1	每个 topic 下每个 partition 保存数据的总量；注意，这是每个 partitions 的上限，因此这个数值乘以 partitions 的个数就是每个 topic 保存的

		数据总量。同时注意：如果 log.retention.hours 和 log.retention.bytes 都设置了，则超过了任何一个限制都会造成删除一个段文件。 注意，这项设置可以由每个 topic 设置时进行覆盖。 查看 the per-topic configuration section
log.retention.check.interval.ms	5 minutes	检查日志分段文件的间隔时间，以确定是否文件属性是否到达删除要求。
log.cleaner.enable	false	当这个属性设置为 false 时，一旦日志的保存时间或者大小达到上限时，就会被删除；如果设置为 true，则当保存属性达到上限时，就会进行 log compaction。
log.cleaner.threads	1	进行日志压缩的线程数
log.cleaner.io.max.bytes.per.second	None	进行 log compaction 时，log cleaner 可以拥有的最大 I/O 数目。这项设置限制了 cleaner，以避免干扰活动的请求服务。
log.cleaner.io.buffer.size	500*1024*1024	log cleaner 清除过程中针对日志进行索引化以及精简化所用到的缓存大小。最好设置大点，以提供充足的内存。
log.cleaner.io.buffer.load.factor	512*1024	进行 log cleaning 时所需要的 I/O chunk 尺寸。

		你不需要更改这项设置。
log.cleaner.io.buffer.load.factor	0.9	log cleaning 中所使用的 hash 表的负载因子；你不需要更改这个选项。
log.cleaner.backoff.ms	15000	进行日志是否清理检查的时间间隔
log.cleaner.min.cleanable.ratio	0.5	这项配置控制 log compactor 试图清理日志的频率（假定 log compaction 是打开的）。默认避免清理压缩超过 50%的日志。这个比率绑定了备份日志所消耗的最大空间（50%的日志备份时压缩率为 50%）。更高的比率则意味着浪费消耗更少，也就可以更有效的清理更多的空间。这项设置在每个 topic 设置中可以覆盖。查看 the per-topic configuration section。
log.cleaner.delete.retention.ms	1day	保存时间；保存压缩日志的最长时间；也是客户端消费消息的最长时间，荣 log.retention.minutes 的区别在于一个控制未压缩数据，一个控制压缩后的数据；会被 topic 创建时的指定时间覆盖。
log.index.size.max.bytes	10*1024*1024	每个 log segment 的最大尺寸。注意，如果 log 尺寸达到这个数值，即使尺寸没有超过 log.segment.bytes 限制，也需要产生新的 log segment。

log.index.interval.bytes	4096	当执行一次 fetch 后，需要一定的空间扫描最近的 offset，设置的越大越好，一般使用默认值就可以
log.flush.interval.messages	Long.MaxValue	log 文件"sync"到磁盘之前累积的消息条数。因为磁盘 IO 操作是一个慢操作，但又是一个"数据可靠性"的必要手段，所以检查是否需要固化到硬盘的时间间隔。需要在"数据可靠性"与"性能"之间做必要的权衡，如果此值过大，将会导致每次"发 sync"的时间过长（IO 阻塞），如果此值过小，将会导致"fsync"的时间较长（IO 阻塞），如果此值过小，将会导致"发 sync"的次数较多，这也就意味着整体的 client 请求有一定的延迟，物理 server 故障，将会导致没有 fsync 的消息丢失。
log.flush.scheduler.interval.ms	Long.MaxValue	检查是否需要 fsync 的时间间隔
log.flush.interval.ms	Long.MaxValue	仅仅通过 interval 来控制消息的磁盘写入时机，是不足的，这个数用来控制"fsync"的时间间隔，如果消息量始终没有达到固化到磁盘的消息数，但是离上次磁盘同步的时间间隔达到阈值，也将触发磁盘同步。
log.delete.delay.ms	60000	文件在索引中清除后的保留时间，一般不需要修改

auto.create.topics.enable	true	是否允许自动创建 topic。如果是真的，则 produce 或者 fetch 不存在的 topic 时，会自动创建这个 topic。否则需要使用命令行创建 topic
controller.socket.timeout.ms	30000	partition 管理控制器进行备份时，socket 的超时时间。
controller.message.queue.size	Int.MaxValue	controller-to-broker-channles 的 buffer 尺寸
default.replication.factor	1	默认备份份数，仅指自动创建的 topics
replica.lag.time.max.ms	10000	如果一个 follower 在这个时间内没有发送 fetch 请求，leader 将从 ISR 重移除这个 follower，并认为这个 follower 已经挂了
replica.lag.max.messages	4000	如果一个 replica 没有备份的条数超过这个数值，则 leader 将移除这个 follower，并认为这个 follower 已经挂了
replica.socket.timeout.ms	30*1000	leader 备份数据时的 socket 网络请求的超时时间
replica.socket.receive.buffer.bytes	64*1024	备份时向 leader 发送网络请求时的 socket receive buffer

replica.fetch.max.bytes	1024*1024	备份时每次 fetch 的最大值
replica.fetch.min.bytes	500	leader 发出备份请求时，数据到达 leader 的最长等待时间
replica.fetch.min.bytes	1	备份时每次 fetch 之后回应的最小尺寸
num.replica.fetchers	1	从 leader 备份数据的线程数
replica.high.watermark.checkpoint.interval.ms	5000	每个 replica 检查是否将最高水位进行固化的频率
fetch.purgatory.purge.interval.requests	1000	fetch 请求清除时的清除间隔
producer.purgatory.purge.interval.requests	1000	producer 请求清除时的清除间隔
zookeeper.session.timeout.ms	6000	zookeeper 会话超时时间。
zookeeper.connection.timeout.ms	6000	客户端等待和 zookeeper 建立连接的最大时间
zookeeper.sync.time.ms	2000	zk follower 落后于 zk leader 的最长时间
controlled.shutdown.enable	true	是否能够控制 broker 的关闭。如果能够，broker 将可以移动所有 leaders 到其他的 broker 上，在关闭之前。这减少了不可用性在关机过程中。
controlled.shutdown.max.retries	3	在执行不彻底的关机之前，可以成功执行关机的

		命令数。
controlled.shutdown.retry.backoff.ms	5000	在关机之间的 backoff 时间
auto.leader.rebalance.enable	true	如果这是 true，控制者将会自动平衡 brokers 对于 partitions 的 leadership
leader.imbalance.per.broker.percentage	10	每个 broker 所允许的 leader 最大不平衡比率
leader.imbalance.check.interval.seconds	300	检查 leader 不平衡的频率
offset.metadata.max.bytes	4096	允许客户端保存他们 offsets 的最大个数
max.connections.per.ip	Int.MaxValue	每个 ip 地址上每个 broker 可以被连接的最大数目
max.connections.per.ip.overrides		每个 ip 或者 hostname 默认的连接的最大覆盖
connections.max.idle.ms	600000	空连接的超时限制
log.roll.jitter.{ms,hours}	0	从 logRollTimeMillis 抽离的 jitter 最大数目
num.recovery.threads.per.data.dir	1	每个数据目录用来日志恢复的线程数目
unclean.leader.election.enable	true	指明了是否能够使不在 ISR 中 replicas 设置用来作为 leader

delete.topic.enable	false	能够删除 topic
offsets.topic.num.partitions	50	The number of partitions for the offset commit topic. Since changing this after deployment is currently unsupported, we recommend using a higher setting for production (e.g., 100-200).
offsets.topic.retention.minutes	1440	存在时间超过这个时间限制的 offsets 都将被标记为待删除
offsets.retention.check.interval.ms	600000	offset 管理器检查陈旧 offsets 的频率
offsets.topic.replication.factor	3	topic 的 offset 的备份份数。建议设置更高的数字保证更高的可用性
offset.topic.segment.bytes	104857600	offsets topic 的 segment 尺寸。
offsets.load.buffer.size	5242880	这项设置与批量尺寸相关，当从 offsets segment 中读取时使用。
offsets.commit.required.acks	-1	在 offset commit 可以接受之前，需要设置确认的数目，一般不需要更改

Property	Default	Server Default Property	Description
cleanup.policy	delete	log.cleanup.policy	要么是”delete“要么是”compact“； 这个字符串指明了针对旧日志部分的利用方式；默认方式（"delete"）将会丢弃旧的部分当他们的回收时间或者尺寸限制到达时。”compact“将会进行日志压缩
delete.retention.ms	86400000 (24 hours)	log.cleaner.delete.retention.ms	对于压缩日志保留的最长时间，也是客户端消费消息的最长时间，通 log.retention.minutes 的区别在于一个控制未压缩数据，一个控制压缩后的数据。此项配置可以在 topic 创建时的置顶参数覆盖
flush.messages	None	log.flush.interval.messages	此项配置指定时间间隔：强制进行 fsync 日志。例如，如果这个选项设置为 1，那么每条消息之后都需要进行 fsync，如果设置为 5，则每 5 条消息就需要进行一次 fsync。一般来说，建议你不要设置这个值。此参数的设置,需要在"数据可靠性"与"性能"之间做必要的权衡.如果此值过大,将会导致每次"fsync"的时间较长(IO 阻塞),如果此值过小,将会导致"fsync"的次数较多,这也意味着整体的 client 请求有一定的延迟.物理 server 故障,将会导

			致没有 fsync 的消息丢失.
flush.ms	None	log.flush.interval.ms	此项配置用来置顶强制进行 fsync 日志到磁盘的时间间隔；例如，如果设置为 1000，那么每 1000ms 就需要进行一次 fsync。一般不建议使用这个选项
index.interval.bytes	4096	log.index.interval.bytes	默认设置保证了我们每 4096 个字节就对消息添加一个索引，更多的索引使得阅读的消息更加靠近，但是索引规模却会由此增大；一般不需要改变这个选项
max.message.bytes	1000000	max.message.bytes	kafka 追加消息的最大尺寸。注意如果你增大这个尺寸，你也必须增大你 consumer 的 fetch 尺寸，这样 consumer 才能 fetch 到这些最大尺寸的消息。
min.cleanable.dirty.ratio	0.5	min.cleanable.dirty.ratio	此项配置控制 log 压缩器试图进行清除日志的频率。默认情况下，将避免清除压缩率超过 50%的日志。这个比率避免了最大的空间浪费
min.insync.replicas	1	min.insync.replicas	当 producer 设置 request.required.acks 为-1 时，min.insync.replicas 指定 replicas 的最

			小数目（必须确认每一个 repica 的写数据都是成功的），如果这个数目没有达到，producer 会产生异常。
retention.bytes	None	log.retention.bytes	如果使用“delete”的 retention 策略，这项配置就是指在删除日志之前，日志所能达到的最大尺寸。默认情况下，没有尺寸限制而只有时间限制
retention.ms	7 days	log.retention.minutes	如果使用“delete”的 retention 策略，这项配置就是指删除日志前日志保存的时间。
segment.bytes	1GB	log.segment.bytes	kafka 中 log 日志是分成一块块存储的，此配置是指 log 日志划分成块的大小
segment.index.bytes	10MB	log.index.size.max.bytes	此配置是有关 offsets 和文件位置之间映射的索引文件的大小；一般不需要修改这个配置
segment.ms	7 days	log.roll.hours	即使 log 的分块文件没有达到需要删除、压缩的大小，一旦 log 的时间达到这个上限，就会强制新建一个 log 分块文件
segment.jitter.ms	0	log.roll.jitter.{ms,hours}	The maximum jitter to subtract from logRollTimeMillis.

Consumer Configs：

Property	Default	Description
group.id		用来唯一标识 consumer 进程所在组的字符串，如果设置同样的 group id，表示这些 processes 都是属于同一个 consumer group
zookeeper.connect		指定 zookeeper 的连接的字符串，格式是 hostname：port，此处 host 和 port 都是 zookeeper server 的 host 和 port，为避免某个 zookeeper 机器宕机之后失联，你可以指定多个 hostname：port，使用逗号作为分隔： hostname1：port1，hostname2：port2，hostname3：port3 可以在 zookeeper 连接字符串中加入 zookeeper 的 chroot 路径，此路径用于存放他自己的数据，方式： hostname1：port1，hostname2：port2，hostname3：port3/chroot/path
consumer.id	null	不需要设置，一般自动产生

socket.timeout.ms	30*100	网络请求的超时限制。真实的超时限制是 max.fetch.wait+socket.timeout.ms
socket.receive.buffer.bytes	64*1024	socket 用于接收网络请求的缓存大小
fetch.message.max.bytes	1024*1024	每次 fetch 请求中，针对每次 fetch 消息的最大字节数。这些字节将会督导用于每个 partition 的内存中，因此，此设置将会控制 consumer 所使用的 memory 大小。这个 fetch 请求尺寸必须至少和 server 允许的最大消息尺寸相等，否则，producer 可能发送的消息尺寸大于 consumer 所能消耗的尺寸。
num.consumer.fetchers	1	用于 fetch 数据的 fetcher 线程数
auto.commit.enable	true	如果为真，consumer 所 fetch 的消息的 offset 将会自动的同步到 zookeeper。这项提交的 offset 将在进程挂掉时，由新的 consumer 使用
auto.commit.interval.ms	60*1000	consumer 向 zookeeper 提交 offset 的频率，单位是秒
queued.max.message.chunks	2	用于缓存消息的最大数目，以供 consumption。每个 chunk 必须和 fetch.message.max.bytes 相同
rebalance.max.retries	4	当新的 consumer 加入到 consumer group 时，consumers 集合试图重新平衡分配到每个 consumer 的 partitions 数目。如果 consumers 集合改变了，当分配正在执行时，这个重新平衡会失败并重入

fetch.min.bytes	1	每次 fetch 请求时，server 应该返回的最小字节数。如果没有足够的数据返回，请求会等待，直到足够的数据才会返回。
fetch.wait.max.ms	100	如果没有足够的数据能够满足 fetch.min.bytes，则此项配置是指在应答 fetch 请求之前，server 会阻塞的最大时间。
rebalance.backoff.ms	2000	在重试 reblance 之前 backoff 时间
refresh.leader.backoff.ms	200	在试图确定某个 partition 的 leader 是否失去他的 leader 地位之前，需要等待的 backoff 时间
auto.offset.reset	largest	zookeeper 中没有初始化的 offset 时，如果 offset 是以下值的回应： smallest：自动复位 offset 为 smallest 的 offset largest：自动复位 offset 为 largest 的 offset anything else：向 consumer 抛出异常
consumer.timeout.ms	-1	如果没有消息可用，即使等待特定的时间之后也没有，则抛出超时异常
exclude.internal.topics	true	是否将内部 topics 的消息暴露给 consumer
paritition.assignment.strategy	range	选择向 consumer 流分配 partitions 的策略，可选值：range，roundrobin
client.id	group id value	是用户特定的字符串，用来在每次请求中帮助跟踪调用。它应该可以逻辑上确认产生这个请求的应用

zookeeper.session.timeout.ms	6000	zookeeper 会话的超时限制。如果 consumer 在这段时间内没有向 zookeeper 发送心跳信息，则它会被认为挂掉了，并且 reblance 将会产生
zookeeper.connection.timeout.ms	6000	客户端在建立通 zookeeper 连接中的最大等待时间
zookeeper.sync.time.ms	2000	ZK follower 可以落后 ZK leader 的最大时间
offsets.storage	zookeeper	用于存放 offsets 的地点： zookeeper 或者 kafka
offset.channel.backoff.ms	1000	重新连接 offsets channel 或者是重试失败的 offset 的 fetch/commit 请求的 backoff 时间
offsets.channel.socket.timeout.ms	10000	当读取 offset 的 fetch/commit 请求回应的 socket 超时限制。此超时限制是被 consumerMetadata 请求用来请求 offset 管理
offsets.commit.max.retries	5	重试 offset commit 的次数。这个重试只应用于 offset commits 在 shut-down 之间。他
dual.commit.enabled	true	如果使用“kafka”作为 offsets.storage，你可以二次提交 offset 到 zookeeper(还有一次是提交到 kafka）。在 zookeeper-based 的 offset storage 到 kafka-based 的 offset storage 迁移时，这是必须的。对任意给定的 consumer group 来说，比较安全的建议是当完成迁移之后就关闭这个选项
partition.assignment.strategy	range	在“range”和“roundrobin”策略之间选择一种作为分配 partitions 给 consumer 数据流的策略； 循环的 partition 分配器分配所有可

		用的 partitions 以及所有可用 consumer 线程。它会将 partition 循环的分配到 consumer 线程上。如果所有 consumer 实例的订阅都是确定的，则 partitions 的划分是确定的分布。循环分配策略只有在以下条件满足时才可以：（1）每个 topic 在每个 consumer 实力上都有同样数量的数据流。（2）订阅的 topic 的集合对于 consumer group 中每个 consumer 实例来说都是确定的。

Producer Configs

Property	Default	Description
metadata.broker.list		服务于 bootstrapping。producer 仅用来获取 metadata（topics，partitions，replicas）。发送实际数据的 socket 连接将基于返回的 metadata 数据信息而建立。格式是： host1：port1，host2：port2 这个列表可以是 brokers 的子列表或者是一个指向 brokers 的 VIP
request.required.acks	0	此配置是表明当一次 produce 请求被认为完

		成时的确认值。特别是，多少个其他 brokers 必须已经提交了数据到他们的 log 并且向他们的 leader 确认了这些信息。典型的值包括： 0： 表示 producer 从来不等待来自 broker 的确认信息（和 0.7 一样的行为）。这个选择提供了最小的时延但同时风险最大（因为当 server 宕机时，数据将会丢失）。 1：表示获得 leader replica 已经接收了数据的确认信息。这个选择时延较小同时确保了 server 确认接收成功。 -1：producer 会获得所有同步 replicas 都收到数据的确认。同时时延最大，然而，这种方式并没有完全消除丢失消息的风险，因为同步 replicas 的数量可能是 1.如果你想确保某些 replicas 接收到数据，那么你应该在 topic-level 设置中选项 min.insync.replicas 设置一下。请阅读一下设计文档，可以获得更深入的讨论。
request.timeout.ms	10000	broker 尽力实现 request.required.acks 需求时的等待时间，否则会发送错误到客户端
producer.type	sync	此选项置顶了消息是否在后台线程中异步发

		送。正确的值： （1） async： 异步发送 （2） sync： 同步发送 通过将 producer 设置为异步，我们可以批量处理请求（有利于提高吞吐率）但是这也就造成了客户端机器丢掉未发送数据的可能性
serializer.class	kafka.serializer.DefaultEncoder	消息的序列化类别。默认编码器输入一个字节 byte[]，然后返回相同的字节 byte[]
key.serializer.class		关键字的序列化类。如果没给与这项，默认情况是和消息一致
partitioner.class	kafka.producer.DefaultPartitioner	partitioner 类，用于在 subtopics 之间划分消息。默认 partitioner 基于 key 的 hash 表
compression.codec	none	此项参数可以设置压缩数据的 codec，可选 codec 为：“none”， “gzip”， “snappy”
compressed.topics	null	此项参数可以设置某些特定的 topics 是否进行压缩。如果压缩 codec 是 NoCompressCodec 之外的 codec，则对指定的 topics 数据应用这些 codec。如果压缩 topics 列表是空，则将特定的压缩 codec 应

		用于所有 topics。如果压缩的 codec 是 NoCompressionCodec，压缩对所有 topics 军不可用。
message.send.max.retries	3	此项参数将使 producer 自动重试失败的发送请求。此项参数将置顶重试的次数。注意：设定非 0 值将导致重复某些网络错误：引起一条发送并引起确认丢失
retry.backoff.ms	100	在每次重试之前，producer 会更新相关 topic 的 metadata，以此进行查看新的 leader 是否分配好了。因为 leader 的选择需要一点时间，此选项指定更新 metadata 之前 producer 需要等待的时间。
topic.metadata.refresh.interval.ms	600*1000	producer 一般会在某些失败的情况下（partition missing，leader 不可用等）更新 topic 的 metadata。他将会规律的循环。如果你设置为负值，metadata 只有在失败的情况下才更新。如果设置为 0，metadata 会在每次消息发送后就会更新（不建议这种选择，系统消耗太大）。重要提示： 更新是有在消息发送后才会发生，因此，如果 producer 从来不发送消息，则 metadata 从来也不会更新。

queue.buffering.max.ms	5000	当应用 async 模式时，用户缓存数据的最大时间间隔。例如，设置为 100 时，将会批量处理 100ms 之内消息。这将改善吞吐率，但是会增加由于缓存产生的延迟。
queue.buffering.max.messages	10000	当使用 async 模式时，在在 producer 必须被阻塞或者数据必须丢失之前，可以缓存到队列中的未发送的最大消息条数
batch.num.messages	200	使用 async 模式时，可以批量处理消息的最大条数。或者消息数目已到达这个上线或者是 queue.buffer.max.ms 到达，producer 才会处理
send.buffer.bytes	100*1024	socket 写缓存尺寸
client.id	“”	这个 client id 是用户特定的字符串，在每次请求中包含用来追踪调用，他应该逻辑上可以确认是那个应用发出了这个请求。

Name	Type	Default	Importance	Description
bootstrap.servers	list		high	用于建立与 kafka 集群连接的 host/port 组。

				数据将会在所有 servers 上均衡加载，不管哪些 server 是指定用于 bootstrapping。这个列表仅仅影响初始化的 hosts（用于发现全部的 servers）。这个列表格式： host1:port1,host2:port2,... 因为这些 server 仅仅是用于初始化的连接，以发现集群所有成员关系（可能会动态的变化），这个列表不需要包含所有的 servers（你可能想要不止一个 server，尽管这样，可能某个 server 宕机了）。如果没有 server 在这个列表出现，则发送数据会一直失败，直到列表可用。
acks	string	1	high	producer 需要 server 接收到数据之后发出的确认接收的信号，此项配置就是指 procuder 需要多少个这样的确认信号。此配置实际上代表了数据备份的可用性。以下设置为常用选项： （1）acks=0： 设置为 0 表示 producer 不需要等待任何确认收到的信息。副本将立即加到 socket buffer 并认为已经发送。没有任何保障可以保证此种情况下 server 已经成功接收数据，同时重试配置不会发生作用（因为客户端不知道是否失败）回馈的 offset 会总是设置为-1； （2）acks=1： 这意味着至少要等待 leader 已经成功将数据写入本地 log，但是并没有等待所有 follower 是否成功写入。这种情况下，如

				果 follower 没有成功备份数据，而此时 leader 又挂掉，则消息会丢失。 （3）acks=all： 这意味着 leader 需要等待所有备份都成功写入日志，这种策略会保证只要有一个备份存活就不会丢失数据。这是最强的保证。 （4）其他的设置，例如 acks=2 也是可以的，这将需要给定的 acks 数量，但是这种策略一般很少用。
buffer.memory	long	33554432	high	producer 可以用来缓存数据的内存大小。如果数据产生速度大于向 broker 发送的速度，producer 会阻塞或者抛出异常，以“block.on.buffer.full”来表明。 这项设置将和 producer 能够使用的总内存相关，但并不是一个硬性的限制，因为不是 producer 使用的所有内存都是用于缓存。一些额外的内存会用于压缩（如果引入压缩机制），同样还有一些用于维护请求。
compression.type	string	none	high	producer 用于压缩数据的压缩类型。默认是无压缩。正确的选项值是 none、gzip、snappy。压缩最好用于批量处理，批量处理消息越多，压缩性能越好。

retries	int	0	high	设置大于 0 的值将使客户端重新发送任何数据，一旦这些数据发送失败。注意，这些重试与客户端接收到发送错误时的重试没有什么不同。允许重试将潜在的改变数据的顺序，如果这两个消息记录都是发送到同一个 partition，则第一个消息失败第二个发送成功，则第二条消息会比第一条消息出现要早。
batch.size	int	16384	medium	producer 将试图批处理消息记录，以减少请求次数。这将改善 client 与 server 之间的性能。这项配置控制默认的批量处理消息字节数。 不会试图处理大于这个字节数的消息字节数。 发送到 brokers 的请求将包含多个批量处理，其中会包含对每个 partition 的一个请求。 较小的批量处理数值比较少用，并且可能降低吞吐量（0 则会仅用批量处理）。较大的批量处理数值将会浪费更多内存空间，这样就需要分配特定批量处理数值的内存大小。
client.id	string		medium	当向 server 发出请求时，这个字符串会发送给 server。目的是能够追踪请求源头，以此来允许 ip/port 许可列表之外的一些应用可以发送信息。这项应用可以设置任意字符串，因为没有任何功能性的目的，除了记录和跟踪
linger.ms	long	0	medium	producer 组将会汇总任何在请求与发送之间到

				达的消息记录一个单独批量的请求。通常来说，这只有在记录产生速度大于发送速度的时候才能发生。然而，在某些条件下，客户端将希望降低请求的数量，甚至降低到中等负载一下。这项设置将通过增加小的延迟来完成--即，不是立即发送一条记录，producer 将会等待给定的延迟时间以允许其他消息记录发送，这些消息记录可以批量处理。这可以认为是 TCP 种 Nagle 的算法类似。这项设置设定了批量处理的更高的延迟边界：一旦我们获得某个 partition 的 batch.size，他将会立即发送而不顾这项设置，然而如果我们获得消息字节数比这项设置要小的多，我们需要“linger”特定的时间以获取更多的消息。 这个设置默认为 0，即没有延迟。设定 linger.ms=5，例如，将会减少请求数目，但是同时会增加 5ms 的延迟。
max.request.size	int	1028576	medium	请求的最大字节数。这也是对最大记录尺寸的有效覆盖。注意：server 具有自己对消息记录尺寸的覆盖，这些尺寸和这个设置不同。此项设置将会限制 producer 每次批量发送请求的数目，以防发出巨量的请求。
receive.buffer.bytes	int	32768	medium	TCP receive 缓存大小，当阅读数据时使用
send.buffer.bytes	int	131072	medium	TCP send 缓存大小，当发送数据时使用

timeout.ms	int	30000	medium	此配置选项控制 server 等待来自 followers 的确认的最大时间。如果确认的请求数目在此时间内没有实现，则会返回一个错误。这个超时限制是以 server 端度量的，没有包含请求的网络延迟
block.on.buffer.full	boolean	true	low	当我们内存缓存用尽时，必须停止接收新消息记录或者抛出错误。默认情况下，这个设置为真，然而某些阻塞可能不值得期待，因此立即抛出错误更好。设置为 false 则会这样：producer 会抛出一个异常错误：BufferExhaustedException， 如果记录已经发送同时缓存已满
metadata.fetch.timeout.ms	long	60000	low	是指我们所获取的一些元素据的第一个时间数据。元素据包含：topic，host，partitions。此项配置是指当等待元素据 fetch 成功完成所需要的时间，否则会跑出异常给客户端。
metadata.max.age.ms	long	300000	low	以微秒为单位的时间，是在我们强制更新 metadata 的时间间隔。即使我们没有看到任何 partition leadership 改变。
metric.reporters	list	[]	low	类的列表，用于衡量指标。实现 MetricReporter 接口，将允许增加一些类，这些类在新的衡量指标产生时就会改变。JmxReporter 总会包含用

				于注册 JMX 统计
metrics.num.samples	int	2	low	用于维护 metrics 的样本数
metrics.sample.window.ms	long	30000	low	metrics 系统维护可配置的样本数量，在一个可修正的 window size。这项配置配置了窗口大小，例如。我们可能在 30s 的期间维护两个样本。当一个窗口推出后，我们会擦除并重写最老的窗口
recoonect.backoff.ms	long	10	low	连接失败时，当我们重新连接时的等待时间。这避免了客户端反复重连
retry.backoff.ms	long	100	low	在试图重试失败的 produce 请求之前的等待时间。避免陷入发送-失败的死循环中。