| **名称** | **说明** | **类型** | **默认值** | **有效值** | **重要性** |
| --- | --- | --- | --- | --- | --- |
| zookeeper.connect | zookeeper集群的地址，可以是多个，多个之间用逗号分割 | string | localhost:2181 | ip1:port1,ip2:port2 | 高 |
| zookeeper.connection.timeout.ms | 客户端在建立通zookeeper连接中的最大等待时间 | int | null | 6000 | 高 |
| zookeeper.session.timeout.ms | ZooKeeper的session的超时时间，如果在这段时间内没有收到ZK的心跳，则会被认为该Kafka server挂掉了。如果把这个值设置得过低可能被误认为挂掉，如果设置得过高，如果真的挂了，则需要很长时间才能被server得知 | int | 6000 |  | 高 |
| zookeeper.sync.time.ms | 一个ZK follower能落后leader的时间 | int | 2000 |  | 高 |
| listeners | 监听列表(以逗号分隔 不同的协议(如plaintext,trace,ssl、不同的IP和端口)),hostname如果设置为0.0.0.0则绑定所有的网卡地址；如果hostname为空则绑定默认的网卡。如果没有配置则默认为java.net.InetAddress.getCanonicalHostName() | string | null | 如：PLAINTEXT://myhost:9092,TRACE://:9091 或 PLAINTEXT://0.0.0.0:9092, | 高 |
| host.name | 。如果设置了它，会仅绑定这个地址。如果没有设置，则会绑定所有的网络接口，并提交一个给ZK。**不推荐使用 只有当listeners没有设置时才有必要使用。** | string | “’ | 如：”localhost” | 高 |
| port | server用来接受client连接的端口。**不推荐使用,使用listeners配置项代替；只有在listeners没有配置时才使用。** | int | 9092 |  | 高 |
| advertised.host.name | 会将hostname通知给生产者和消费者，在多网卡时需要设置该值为另一个ip地址。如果没有设置该值，则返回 配置项host.name设置的值，如果host.name没有设置则返回java.net.InetAddress.getCanonicalHostName()**不推荐使用 只有当advertised.listeners或listeners没有设置时才有必要使用。** | string | null |  | 高 |
| advertised.listeners | 设置不同于listeners配置的监听列表即不同于listeners设置的网卡地址及端口；如果没有配置，会使用listeners的值 | string | null |  | 高 |
| advertised.port | 分发这个端口给所有的producer，consumer和其他broker来建立连接。如果此端口跟server绑定的端口不同，则才有必要设置。**不推荐使用 只有当advertised.listeners或listeners没有设置时才有必要使用。** | int | null |  | 高 |
| auto.create.topics.enable | 是否允许自动创建topic。如果设为true，那么produce，consume或者fetch metadata一个不存在的topic时，就会自动创建一个默认replication factor和partition number的topic。 | boolean | true |  | 高 |
| background.threads | 一些后台任务处理的线程数，例如过期消息文件的删除等，一般情况下不需要去做修改 | int | 10 |  | 高 |
| broker.id | 每一个broker在集群中的唯一表示，要求是正数。当该服务器的IP地址发生改变时，broker.id没有变化，则不会影响consumers的消息情况。 | int | -1 |  | 高 |
| compression.type | 指定topic的压缩类型。除了支持’gzip’, ‘snappy’, ‘lz4’外，还支持”uncompressed(不压缩)”以及producer(由producer来指定) | string | producer |  | 高 |
| delete.topic.enable | 是否启动删除topic。如果设置为false,你在删除topic的时候无法删除，但是会打上一个你将删除该topic的标记，等到你修改这一属性的值为true后重新启动Kafka集群的时候，集群自动将那些标记删除的topic删除掉，对应的log.dirs目录下的topic目录和数据也会被删除。而将这一属性设置为true之后，你就能成功删除你想要删除的topic了 | boolean | false |  | 高 |
| auto.leader.rebalance.enable | 一个后台线程会周期性的自动尝试，为所有的broker的每个partition平衡leadership，使kafka的leader均衡。 | boolean | true |  | 高 |
| leader.imbalance.check.interval.seconds | 检查leader是否均衡的时间间隔(秒) | long | 300 |  | 高 |
| leader.imbalance.per.broker.percentage | 每个broker允许的不平衡的leader的百分比。如果每个broker超过了这个百分比，复制控制器会重新平衡leadership。 | int | 10 |  | 高 |
| log.flush.interval.messages | 数据flush(sync)到硬盘前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 | long | 9223372036854775807 |  | 高 |
| log.flush.interval.ms | 当达到下面的时间(ms)时，执行一次强制的flush操作。interval.ms和interval.messages无论哪个达到，都会flush。 | long | null |  | 高 |
| log.flush.offset.checkpoint.interval.ms | 记录上次把log刷到磁盘的时间点的频率，用来日后的recovery。通常不需要改变 | long | 60000 |  | 高 |
| log.flush.scheduler.interval.ms | 检查是否需要固化到硬盘的时间间隔 | long | 9223372036854775807 |  | 高 |
| log.retention.bytes | topic每个分区的最大文件大小，一个topic的大小限制 = 分区数\*log.retention.bytes。-1没有大小限log.retention.bytes和log.retention.minutes任意一个达到要求，都会执行删除，会被topic创建时的指定参数覆盖 | loong | -1 |  | 高 |
| log.retention.hours | 日志保存时间，默认为7天（168小时）。超过这个时间会根据policy处理数据。bytes和minutes无论哪个先达到都会触发 | int | 168 |  | 高 |
| log.retention.minutes | 数据存储的最大时间超过这个时间会根据log.cleanup.policy设置的策略处理数据，也就是消费端能够多久去消费数据 |  |  |  |  |
| log.retention.bytes和log.retention.minutes任意一个达到要求，都会执行删除，会被topic创建时的指定参数覆盖 | int | null |  | 高 |  |
| log.roll.hous | 当达到下面时间，会强制新建一个segment。这个参数会在日志segment没有达到log.segment.bytes设置的大小，也会强制新建一个segment会 | int | 168 |  | 高 |
| log.roll.jitter.{ms,hours} | 从logRollTimeMillis抽离的jitter最大数目 | int | 0 |  | 高 |
| log.segment.bytes | 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 | long | 1G=1024\*1024\*1024 |  | 高 |
| log.segment.delet.delay.ms | 删除文件系统上文件的等待时间，默认是1分钟 | long | 6000 |  | 高 |
| message.max.bytes | 表示一个服务器能够接收处理的消息的最大字节数，注意这个值producer和consumer必须设置一致，且不要大于fetch.message.max.bytes属性的值该值默认是1000012字节，大概900KB |  |  |  |  |
| int | 1000012 |  | 高 |  |  |
| min.insync.replicas | 该属性规定了最小的ISR数。当producer设置request.required.acks为all或-1时，指定副本(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。 | int | 1 |  | 高 |
| num.io.threads | 服务器用来处理请求的I/O线程的数目；这个线程数目至少要等于硬盘的个数。 | int | 8 |  | 高 |
| num.network.threads | 服务器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 | int | 3 |  | 高 |
| num.recovery.threads.per.data.dir | 每数据目录用于日志恢复启动和关闭冲洗时的线程数量 | int | 1 |  | 高 |
| num.replica.fetchers | 从leader进行复制消息的线程数，增大这个数值会增加follower的IO | int | 1 |  | 高 |
| offset.metadata.max.bytes | 允许client(消费者)保存它们元数据(offset)的最大的数据量 | int | 4096(4kb) |  |  |
| offsets.commit.required.acks | 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 | int | -1 |  | 高 |
| offsets.commit.timeout.ms | offset commit会延迟直至此超时或所需的副本数都收到offset commit，这类似于producer请求的超时 | int | 5000 |  | 高 |
| offsets.load.buffer.size | 此设置对应于offset manager在读取缓存offset segment的批量大小（以字节为单位). | int | 5242880 |  | 高 |
| offsets.retention.check.interval.ms | offset管理器检查陈旧offsets的频率 | long | 600000(10分钟) |  | 高 |
| offsets.topic.num.partitions | 偏移的提交topic的分区数目。 由于目前不支持部署之后改变，我们建议您使用生产较高的设置（例如，100-200） | int | 50 |  | 高 |
| offsets.topic.replication.factor | 复制因子的offset提交topic。较高的设置（例如三个或四个），建议以确保更高的可用性。如果offset topic创建时，broker比复制因子少，offset topic将以较少的副本创建。 | short | 3 |  | 高 |
| offsets.topic.segment.bytes | offset topic的Segment大小。因为它使用压缩的topic，所有Sgment的大小应该保持小一点，以促进更快的日志压实和负载 | int | 104857600 |  | 高 |
| queued.max.requests | 在网络线程(network threads)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。若是等待IO的请求超过这个数值，那么会停止接受外部消息 | int | 500 |  | 高 |
| quota.consumer.default | 以clientid或consumer group区分的consumer端每秒可以抓取的最大byte | long | 9223372036854775807 |  | 高 |
| quota.producer.default | producer端每秒可以产生的最大byte | long | 9223372036854775807 |  | 高 |
| replica.fetch.max.bytes | replicas每次获取数据的最大字节数 | int | 1048576 |  | 高 |
| replica.fetch.min.bytes | fetch的最小数据尺寸,如果leader中尚未同步的数据不足此值,将会阻塞,直到满足条件 | int | 1 |  | 高 |
| replica.fetch.wait.max.ms | replicas同leader之间通信的最大等待时间，失败了会重试。这个值须小于replica.lag.time.max.ms，以防止低吞吐量主题ISR频繁收缩 | int | 500 |  | 高 |
| replica.high.watermark.checkpoint.interval.ms | 每一个replica存储自己的high watermark到磁盘的频率，用来日后的recovery | int | 5000 |  | 高 |
| replica.socket.timeout.ms | 复制数据过程中，replica发送给leader的网络请求的socket超时时间,至少等于replica.fetch.wait.max.ms | int | 30000 |  | 高 |
| replica.socket.receive.buffer.bytes | 复制过程leader接受请求的buffer大小 | int | 65536(64\*1024) |  | 高 |
| replica.lag.time.max.ms | replicas响应partition leader的最长等待时间，若是超过这个时间，就将replicas列入ISR(in-sync replicas)，并认为它是死的，不会再加入管理中 | long | 10000 |  | 高 |
| replica.lag.max.messages | 如果follower落后与leader太多,将会认为此follower[或者说partition relicas]已经失效。 通常,在follower与leader通讯时,因为网络延迟或者链接断开,总会导致replicas中消息同步滞后如果消息之后太多,leader将认为此follower网络延迟较大或者消息吞吐能力有限,将会把此replicas迁移到其他follower中.在broker数量较少,或者网络不足的环境中,建议提高此值. | int | 4000 |  | 高 |
| request.timeout.ms | producer等待响应的最长时间，如果超时将重发几次，最终报错 | int | 30000 |  | 高 |
| socket.receive.buffer.bytes | socket用于接收网络请求的缓存大小 | int | 102400 |  | 高 |
| socket.request.max.bytes | server能接受的请求的最大的大小，这是为了防止server跑光内存，不能大于Java堆的大小。 | int | 104857600(100\*1024\*1024) |  | 高 |
| socket.send.buffer.bytes | server端用来处理socket连接的SO\_SNDBUFF缓冲大小 | int | 102400 |  | 高 |
| controller.socket.timeout.ms | partition管理控制器进行备份时，socket的超时时间 | int | 30000 |  | 高 |
| controller.message.queue.size | partition leader与replicas数据同步时,消息的队列大小 | int | 10 |  | 高 |
| num.partitions | 每个topic的分区个数，若是在topic创建时候没有指定的话会被topic创建时的指定参数覆盖 | int | 1 | 推荐设为8 | 高 |
| log.index.interval.bytes | 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，但是也更耗内存一般使用默认值就可以 | int | 4096 |  | 中 |
| log.index.size.max.bytes | 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。 | int | 10485760 |  | 中 |
| fetch.purgatory.purge.interval.requests | 非立即答复请求放入purgatory中，当到达或超出interval时认为request complete | int | 1000 |  | 中 |
| producer.purgatory.purge.interval.requests | producer请求清除时间 | int | 1000 |  | 中 |
| default.replication.factor | 一个topic ，默认分区的replication个数 ，不能大于集群中broker的个数。 | int | 1 |  | 中 |
| group.max.session.timeout.ms | 注册consumer允许的最大超时时间 | int | 300000 |  | 中 |
| group.min.session.timeout.ms | 注册consumer允许的最小超时时间 | int | 6000 |  | 中 |
| inter.broker.protocol.version | broker协议版本 | string | 0.10.0 |  | 中 |
| log.cleaner.backoff.ms | 检查log是否需要clean的时间间隔 | long | 15000 |  | 中 |
| log.cleaner.dedupe.buffer.size | 日志压缩去重时候的缓存空间，在空间允许的情况下，越大越好 | long | 134217728 |  | 中 |
| log.cleaner.delete.retention.ms | 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，同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。 | long | 86400000(一天) |  | 中 |
| log.cleaner.enable | 是否启动压缩日志,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，就会进行压缩 | boolean | false |  | 中 |
| log.cleaner.threads | 日志压缩运行的线程数 | int | 1 |  | 中 |
| log.cleaner.io.buffer.load.factor | 日志清理中hash表的扩大因子，一般不需要修改 | double | 0.9 |  | 中 |
| log.cleaner.io.buffer.size | 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 | int | 524288 |  | 中 |
| log.cleaner.io.max.bytes.per.second | 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。 | double | 1.7976931348623157E308 |  | 中 |
| log.cleaner.min.cleanable.ratio | 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 | double | 0.5 |  | 中 |
| log.preallocate | 是否预创建新文件，windows推荐使用 | boolean | false |  | 中 |
| log.retention.check.interval.ms | 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。 | long | 300000 |  | 中 |
| max.connections.per.ip | 一个broker允许从每个ip地址连接的最大数目 | int | 2147483647=Int.MaxValue |  | 中 |
| max.connections.per.ip.overrides | 每个IP或主机名覆盖连接的默认最大数量 | string | “” |  | 中 |
| replica.fetch.backoff.ms | 复制数据时失败等待时间 | int | 1000 |  | 中 |
| reserved.broker.max.id | broker可以使用的最大ID值 | int | 1000 |  |  |

**配置详细说明**