配置管理

kafka-configs.sh 脚本是专门用来对配置进行操作的，这里的操作是指在运行状态下修改原有的配置，如此可以达到动态变更的目的。kafka-configs.sh 脚本包含变更配置 alter 和查看配置 describe 这两种指令类型。同使用 kafka-topics.sh 脚本变更配置的原则一样，增、删、改的行为都可以看作变更操作，不过 kafka-configs.sh 脚本不仅可以支持操作主题相关的配置，还可以支持操作 broker、用户和客户端这3个类型的配置。

kafka-configs.sh 脚本使用 entity-type 参数来指定操作配置的类型，并且使用 entity-name 参数来指定操作配置的名称。比如查看主题 topic-config 的配置可以按如下方式执行：

```
bin/kafka-configs.sh --zookeeper
localhost:2181/kafka --describe --entity-type
topics --entity-name topic-config
```

--describe 指定了查看配置的指令动作，--entity-type 指定了查看配置的实体类型，--entity-name 指定了查看配置的实体名称。entity-type 只可以配置4个值：topics、brokers 、clients 和 users，entity-type 与 entity-name 的对应关系如下表所示。

entity-type 的释义	entity-name 的释义
主题类型的配置，取值为 topics	指定主题的名称
broker 类型的配置，取值为 brokers	指定 brokerId 值，即 broker 中 broker.id 参数配置的值
客户端类型的配置，取值为 clients	指定 clientId 值，即 KafkaProducer 或 KafkaConsumer 的 client.id 参数配置的值

用户类型的配置，取值为 users	指定用户名

使用 alter 指令变更配置时，需要配合 add-config 和 delete-config 这两个参数一起使用。add-config 参数用来实现配置的增、改，即覆盖原有的配置；delete-config 参数用来实现配置的删，即删除被覆盖的配置以恢复默认值。

下面的示例演示了 add-config 参数的用法，覆盖了主题 topic-config 的两个配置 cleanup.policy 和 max.message.bytes（示例执行之前主题 topic-config 无任何被覆盖的配置）：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
configs.sh --zookeeper localhost:2181/ kafka --
alter --entity-type topics --entity-name topic-
config --add-config
cleanup.policy=compact,max.message.bytes=10000
Completed Updating config for entity: topic
'topic-config'.

[root@node1 kafka_2.11-2.0.0]# bin/kafka-
configs.sh --zookeeper localhost:2181/ kafka --
describe --entity-type topics --entity-name
topic-config
Configs for topic 'topic-config' are
max.message.bytes=10000,cleanup.policy= compact

[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
describe --topic topic-config --topics-with-
overrides
Topic:topic-config  PartitionCount:3
ReplicationFactor:1
Configs:max.message.bytes=10000,cleanup.policy=co
mpact
```

上面示例中还使用了两种方式来查看主题 topic-config 中配置信息，注意比较这两者之间的差别。

使用 delete-config 参数删除配置时，同 add-config 参数一样支持多个配置的操作，多个配置之间用逗号“,”分隔，下面的示例中演示了如何删除上面刚刚增加的主题配置：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
configs.sh --zookeeper localhost:2181/ kafka --
alter --entity-type topics --entity-name topic-
config --delete-config
cleanup.policy,max.message.bytes
Completed Updating config for entity: topic
'topic-config'.

[root@node1 kafka_2.11-2.0.0]# bin/kafka-
configs.sh --zookeeper localhost:2181/ kafka --
describe --entity-type topics --entity-name
topic-config
Configs for topic 'topic-config' are
```

使用 kafka-configs.sh 脚本来变更（alter）配置时，会在 ZooKeeper 中创建一个命名形式为/config/<entity-type>/<entity-name>的节点，并将变更的配置写入这个节点，比如对于主题 topic-config 而言，对应的节点名称为/config/topics/topic-config，节点中的数据内容为：

```
[zk: localhost:2181/kafka (CONNECTED) 1] get
/config/topics/topic-config
{"version":1,"config":
{"cleanup.policy":"compact","max.message.bytes":"
10000"}}
```

可以推导出节点内容的数据格式为：

```
{"version":1,"config":{<property-name>:<property-
value>}}
```

其中 property-name 代表属性名，property-value 代表属性值。增加配置实际上是往节点内容中添加属性的键值对，修改配置是在节点内容中修改相应属性的属性值，删除配置是删除相应的属性键值

对。

变更配置时还会在 ZooKeeper 中的/config/changes/节点下创建一个以“config_change_”为前缀的持久顺序节点（PERSISTENT_SEQUENTIAL），节点命名形式可以归纳为/config/changes/config_change_<seqNo>。比如示例中的主题 topic-config 与此对应的节点名称和节点内容如下：

```
[zk: localhost:2181/kafka (CONNECTED) 3] get
      /config/changes/config_change_0000000010
{"version":2,"entity_path":"topics/topic-config"}
```

seqNo 是一个单调递增的10位数字的字符串，不足位则用0补齐。

查看（describe）配置时，就是从/config/<entity-type>/<entity-name>节点中获取相应的数据内容。如果使用 kafka-configs.sh 脚本查看配置信息时没有指定 entity-name 参数的值，则会查看 entity-type 所对应的所有配置信息。示例如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
configs.sh --zookeeper localhost:2181/ kafka --
describe --entity-type topics
Configs for topic 'topic-config' are

cleanup.policy=compact,max.message.bytes=20000
Configs for topic 'topic-create' are
Configs for topic '__consumer_offsets' are

segment.bytes=104857600,cleanup.policy=compact,co
mpression.type=producer
Configs for topic 'topic-demo' are
```

主题端参数

与主题相关的所有配置参数在 broker 层面都有对应参数，比如主题端参数 cleanup. policy 对应 broker 层面的 log.cleanup.policy。如果没有修改过主题的任何配置参数，那么就会使用 broker 端的对应参数作为其默认值。可以在创建主题时覆盖相应参数的默认值，也可以在创建完主题之后变更相应参数的默认值。比如在创建主题的时候没有指定 cleanup.policy 参数的值，那么就使用 log.cleanup.policy 参数所配置的值作为 cleanup.policy 的值。

与主题相关的参数也有很多，由于篇幅限制，在前面的配置变更的示例中难以一一列出所有的参数，但是从配置变更的角度而言，其操作方式都是一样的。为了便于读者查阅，下表列出了主题端参数与 broker 端参数的对照关系。

主题端参数	释 义	
cleanup.policy	日志压缩策略。默认值为 delete，还可以配置为 compact	log.c
compression.type	消息的压缩类型。默认值为 producer，表示保留生产者中所使用的原始压缩类型。还可以配置为 uncompressed、snappy、lz4、gzip	comp
delete.retention.ms	被标识为删除的数据能够保留多久。默认值为86400000，即1天	log.c
file.delete.delay.ms	清理文件之前可以等待多长时间，默认值为60000，即1分钟	log.s
	需要收集多少消息才会将它们强制刷新到	

flush.messages	磁盘，默认值为 Long.MAX_VALUE，即让操作系统来决定。建议不要修改此参数的默认值	log.f
flush.ms	需要等待多久才会将消息强制刷新到磁盘，默认值为 Long.MAX_VALUE，即让操作系统来决定。建议不要修改此参数的默认值	log.f
follower.replication.throttled.replicas	用来配置被限制速率的主题所对应的 follower 副本列表	follo
index.interval.bytes	用来控制添加索引项的频率。每超过这个参数所设置的消息字节数时就可以添加一个新的索引项，默认值为4096	log.i
leader.replication.throttled.replicas	用来配置被限制速率的主题所对应的 leader 副本列表	leade
max.message.bytes	消息的最大字节数，默认值为1000012	mess
message.format.version	消息格式的版本，默认值为 2.0-IV1	log.r
	消息中自带的时间戳与 broker 收到消息时的时间戳之间最大的差值，默认值为	

message.timestamp.difference.max.ms	Long.MAX_VALUE。此参数只有在 meesage.timestamp.type 参数设置为 CreateTime 时才有效	log.r diffe
message.timestamp.type	消息的时间戳类型。默认值为 CreateTime，还可以设置为 LogAppendTime	log.r
min.cleanable.dirty.ratio	日志清理时的最小污浊率，默认值为0.5	log.c
min.compaction.lag.ms	日志再被清理前的最小保留时间，默认值为0	log.c
min.insync.replicas	分区ISR集合中至少要有多少个副本，默认值为1	min.i
preallocate	在创建日志分段的时候是否要预分配空间，默认值为 false	log.p
retention.bytes	分区中所能保留的消息总量，默认值为-1，即没有限制	log.r
retention.ms	使用 delete 的日志清理策略时消息能够保留多长时间，默认值为604800000，即7天。如果设置为-1，则表示没有限	log.r

	制	
segment.bytes	日志分段的最大值，默认值为1073741824，即1GB	log.s
segment.index.bytes	日志分段索引的最大值，默认值为10485760，即10MB	log.i
segment.jitter.ms	滚动日志分段时，在segment.ms 的基础之上增加的随机数，默认为0	log.r
segment.ms	最长多久滚动一次日志分段，默认值为604800000，即7天	log.r
unclean.leader.election.enable	是否可以从非 ISR 集合中选举 leader 副本，默认值为false，如果设置为true，则可能造成数据丢失	uncle