主题与分区

主题和分区是 Kafka 的两个核心概念，前面章节中讲述的生产者和消费者的设计理念所针对的都是主题和分区层面的操作。主题作为消息的归类，可以再细分为一个或多个分区，分区也可以看作对消息的二次归类。分区的划分不仅为 Kafka 提供了可伸缩性、水平扩展的功能，还通过多副本机制来为 Kafka 提供数据冗余以提高数据可靠性。

从 Kafka 的底层实现来说，主题和分区都是逻辑上的概念，分区可以有一至多个副本，每个副本对应一个日志文件，每个日志文件对应一至多个日志分段（LogSegment），每个日志分段还可以细分为索引文件、日志存储文件和快照文件等。不过对于使用 Kafka 进行消息收发的普通用户而言，了解到分区这一层面足以应对大部分的使用场景。本章只针对主题与分区这一层面的内容进行讲解，更底层的内容会在[《图解Kafka之核心原理》(https://juejin.im/book/5c7d270ff265da2d89634e9e)](https://juejin.im/book/5c7d270ff265da2d89634e9e)中进行详述。

主题的管理

主题的管理包括创建主题、查看主题信息、修改主题和删除主题等操作。可以通过 Kafka 提供的 kafka-topics.sh 脚本来执行这些操作，这个脚本位于$KAFKA_HOME/bin/目录下，其核心代码仅有一行，具体如下：

```
exec $(dirname $0)/kafka-run-class.sh
kafka.admin.TopicCommand "$@"
```

可以看到其实质上是调用了 kafka.admin.TopicCommand 类来执行主题管理的操作。

主题的管理并非只有使用 kafka-topics.sh 脚本这一种方式，我们还可以通过 KafkaAdminClient 的方式实现（这种方式实质上是通过发送 CreateTopicsRequest、DeleteTopicsRequest 等请求来实现的），甚至我们还可以通过直接操纵日志文件和 ZooKeeper 节点来实现。下面按照创建主题、查看主题信息、修改主题、删除主题的顺序来介绍其中的操作细节。

创建主题

如果 broker 端配置参数 auto.create.topics.enable 设置为 true（默认值就是 true），那么当生产者向一个尚未创建的主题发送消息时，会自动创建一个分区数为 num.partitions（默认值为 1）、副本因子为 default.replication.factor（默认值为1）的主题。除此之外，当一个消费者开始从未知主题中读取消息时，或者当任意一个客户端向未知主题发送元数据请求时，都会按照配置参数 num.partitions 和 default.replication.factor 的值来创建一个相应的主题。很多时候，这种自动创建主题的行为都是非预期的。除非有特殊应用需求，否则不建议将 auto.create.topics.enable 参数设置为 true，这个参数会增加主题的管理与维护的难度。

更加推荐也更加通用的方式是通过 kafka-topics.sh 脚本来创建主题。在第2节演示消息的生产与消费时就通过这种方式创建了一个分区数为4、副本因子为3的主题 topic-demo。下面通过创建另一个主题 topic-create 来回顾一下这种创建主题的方式，示例如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/kafka --
create --topic topic-create --partitions 4 --
replication-factor 2
Created topic "topic-create". #此为控制台执行的输出结
果
```

上面的示例中创建了一个分区数为4、副本因子为2的主题。示例中的环境是一个包含3个 broker 节点的集群，每个节点的名称和 brokerId 的对照关系如下：

```
node1 brokerId=0
node2 brokerId=1
node3 brokerId=2
```

在执行完脚本之后，Kafka 会在 log.dir 或 log.dirs 参数所配置的目录下创建相应的主题分区，默认情况下这个目录为/tmp/kafka-logs/。我们来查看一下 node1 节点中创建的主题分区，参考如下：

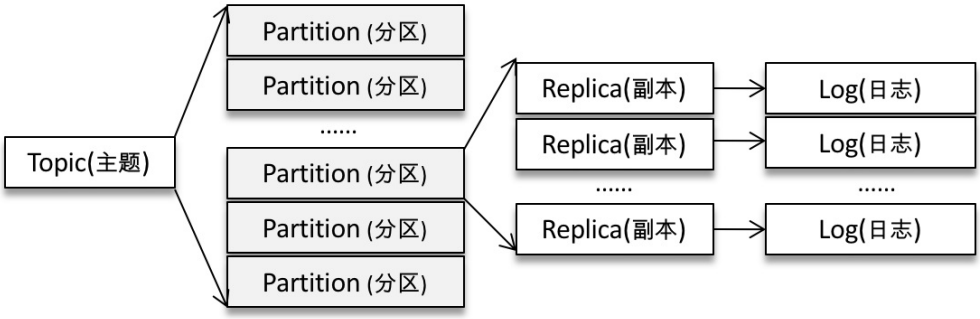
```
[root@node1 kafka_2.11-2.0.0]# ls -al /tmp/kafka-
logs/ | grep topic-create
drwxr-xr-x   2 root root 4096 Sep  8 15:54 topic-
create-0
drwxr-xr-x   2 root root 4096 Sep  8 15:54 topic-
create-1
```

可以看到 node1 节点中创建了2个文件夹 topic-create-0 和 topic-create-1，对应主题 topic-create 的2个分区编号为0和1的分区，命名方式可以概括为-。严谨地说，其实-这类文件夹对应的不是分区，分区同主题一样是一个逻辑的概念而没有物理上的存在。并且这里我们也只是看到了2个分区，而我们创建的是4个分区，其余2个分区被分配到了 node2 和 node3 节点中，参考如下：

```
[root@node2 kafka_2.11-2.0.0]# ls -al /tmp/kafka-logs/ |grep topic-create
drwxr-xr-x   2 root root   4096 Sep  8 15:49 topic-create-1
drwxr-xr-x   2 root root   4096 Sep  8 15:49 topic-create-2
drwxr-xr-x   2 root root   4096 Sep  8 15:49 topic-create-3
[root@node3 kafka_2.11-2.0.0]# ls -al /tmp/kafka-logs/ |grep topic-create
drwxr-xr-x   2 root root 4096 Sep  8 07:54 topic-create-0
drwxr-xr-x   2 root root 4096 Sep  8 07:54 topic-create-2
drwxr-xr-x   2 root root 4096 Sep  8 07:54 topic-create-3
```

三个 broker 节点一共创建了8个文件夹，这个数字8实质上是分区数4与副本因子2的乘积。每个副本（或者更确切地说应该是日志，副本与日志一一对应）才真正对应了一个命名形式如<topic>-<partition>的文件夹。

主题、分区、副本和 Log（日志）的关系如下图所示，主题和分区都是提供给上层用户的抽象，而在副本层面或更加确切地说是 Log 层面才有实际物理上的存在。同一个分区中的多个副本必须分布在不同的 broker 中，这样才能提供有效的数据冗余。对于示例中的分区数为4、副本因子为2、broker 数为3的情况下，按照2、3、3的分区副本个数分配给各个 broker 是最优的选择。再比如在分区数为3、副本因子为3，并且 broker 数同样为3的情况下，分配3、3、3的分区副本个数给各个 broker 是最优的选择，也就是每个 broker 中都拥有所有分区的一个副本。



我们不仅可以通过日志文件的根目录来查看集群中各个 broker 的分区副本的分配情况，还可以通过 ZooKeeper 客户端来获取。当创建一个主题时会在 ZooKeeper 的/brokers/topics/目录下创建一个同名的实节点，该节点中记录了该主题的分区副本分配方案。示例如下：

```
[zk: localhost:2181/kafka(CONNECTED) 2] get
/brokers/topics/topic-create
{"version":1,"partitions":{"2":[1,2],"1":
[0,1],"3":[2,1],"0":[2,0]}}
```

示例数据中的"2":[1,2]表示分区2分配了2个副本，分别在 brokerId 为1和2的 broker 节点中。
回顾一下第2节中提及的知识点：kafka-topics.sh 脚本中的 zookeeper、partitions、replication-factor 和 topic 这4个参数分别代表 ZooKeeper 连接地址、分区数、副本因子和主题名称。另一个 create 参数表示的是创建主题的指令类型，在 kafka-topics.sh 脚本中对应的还有 list、describe、alter 和 delete 这4个同级别的指令类型，每个类型所需要的参数也不尽相同。

还可以通过 describe 指令类型来查看分区副本的分配细节，示例如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/kafka --
describe --topic topic-create
Topic:topic-create  PartitionCount:4
ReplicationFactor:2 Configs:
    Topic: topic-create Partition: 0    Leader: 2
Replicas: 2,0   Isr: 2,0
    Topic: topic-create Partition: 1    Leader: 0
Replicas: 0,1   Isr: 0,1
    Topic: topic-create Partition: 2    Leader: 1
Replicas: 1,2   Isr: 1,2
    Topic: topic-create Partition: 3    Leader: 2
Replicas: 2,1   Isr: 2,1
```

示例中的 Topic 和 Partition 分别表示主题名称和分区号。PartitionCount 表示主题中分区的个数，ReplicationFactor 表示副本因子，而 Configs 表示创建或修改主题时指定的参数配置。Leader 表示分区的 leader 副本所对应的 brokerId，Isr 表示分区的 ISR 集合，Replicas 表示分区的所有的副本分配情况，即AR集合，其中的数字都表示的是 brokerId。

使用 kafka-topics.sh 脚本创建主题的指令格式归纳如下：

```
kafka-topics.sh --zookeeper <String: hosts>
-create --topic [String: topic] --partitions
<Integer: # of partitions> -replication-factor
<Integer: replication factor>
```

到目前为止，创建主题时的分区副本都是按照既定的内部逻辑来进行分配的。kafka-topics.sh 脚本中还提供了一个 replica-assignment 参数来手动指定分区副本的分配方案。replica-assignment 参数的用法归纳如下：

```
--replica-assignment <String:
broker_id_for_part1_replica1:  broker_id_for_
part1_replica2, broker_id_for_part2_replica1:
broker_id_for_part2_replica2, …>
```

这种方式根据分区号的数值大小按照从小到大的顺序进行排列，分区与分区之间用逗号“,”隔开，分区内多个副本用冒号“:”隔开。并且在使用 replica-assignment 参数创建主题时不需要原本必备的 partitions 和 replication-factor 这两个参数。

我们可以通过 replica-assignment 参数来创建一个与主题 topic-create 相同的分配方案的主题 topic-create-same 和不同的分配方案的主题 topic-create-diff，示例如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
create --topic topic-create-same --replica-
assignment 2:0,0:1,1:2,2:1
Created topic "topic-create-same".

[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/kafka --
describe --topic topic-create-same
Topic:topic-create-same PartitionCount:4
ReplicationFactor:2 Configs:
    Topic: topic-create-same    Partition: 0
Leader: 2 Replicas: 2,0 Isr: 2,0
        Topic: topic-create-same    Partition: 1
Leader: 0 Replicas: 0,1 Isr: 0,1
        Topic: topic-create-same    Partition: 2
Leader: 1 Replicas: 1,2 Isr: 1,2
        Topic: topic-create-same    Partition: 3
Leader: 2 Replicas: 2,1 Isr: 2,1
```

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
create --topic topic-create-diff --replica-
assignment 1:2,2:0,0:1,1:0
Created topic "topic-create-diff".

[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
describe --topic topic-create-diff
Topic:topic-create-diff PartitionCount:4
ReplicationFactor:2 Configs:
        Topic: topic-create-diff    Partition: 0
Leader: 1 Replicas: 1,2 Isr: 1,2
        Topic: topic-create-diff    Partition: 1
Leader: 2 Replicas: 2,0 Isr: 2,0
        Topic: topic-create-diff    Partition: 2
Leader: 0 Replicas: 0,1 Isr: 0,1
        Topic: topic-create-diff    Partition: 3
Leader: 1 Replicas: 1,0 Isr: 1,0
```

注意同一个分区内的副本不能有重复，比如指定了0:0,1:1这种，就会报出 AdminCommand- FailedException 异常，示例如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
create --topic topic-create-error --replica-
assignment 0:0,1,1
Error while executing topic command : Partition
replica lists may not contain duplicate entries:
0
[2018-09-09 11:17:02,549] ERROR
kafka.common.AdminCommandFailedException:
Partition replica lists may not contain duplicate
entries: 0 at …(省略若干)
```

如果分区之间所指定的副本数不同，比如0:1,0,1:0这种，就会报出AdminOperationException 异常，示例如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
create --topic topic-create-error --replica-
assignment 0:1,0,1:0
Error while executing topic command : Partition 1
has different replication factor: [I@5e0826e7
[2018-09-09 11:17:15,684] ERROR
kafka.admin.AdminOperationException: Partition 1
has different replication factor: [I@5e0826e7 at
…(省略若干)
```

当然，类似0:1,,0:1,1:0这种企图跳过一个分区的行为也是不被允许的，示例如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka200 --
create --topic topic-create-error --replica-
assignment 0:1,,0:1,1:0
Error while executing topic command : For input
string: ""
[2018-09-09 11:17:27,767] ERROR
java.lang.NumberFormatException: For input
string: "" at …(省略若干)
```

在创建主题时我们还可以通过 config 参数来设置所要创建主题的相关参数，通过这个参数可以覆盖原本的默认配置。在创建主题时可以同时设置多个参数，具体的用法归纳如下：

```
--config <String:name1=value1>  --config
<String:name2=value2>
```

下面的示例使用了 config 参数来创建一个主题 topic-config：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
create --topic topic-config --replication-factor
1 --partitions 1 --config cleanup.policy=compact
--config max.message.bytes=10000
Created topic "topic-config".
```

示例中设置了 cleanup.policy 参数为 compact，以及 max.message.bytes 参数为10000，这两个参数都是主题端的配置，我们再次通过 describe 指令来查看所创建的主题信息：

```
 [root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
describe --topic topic-config
Topic:topic-config  PartitionCount:1
ReplicationFactor:1
Configs:cleanup.policy=compact,max.message.bytes=
10000
    Topic: topic-config Partition: 0    Leader: 0
Replicas: 0 Isr: 0
```

可以看到 Configs 一栏中包含了创建时所设置的参数。我们还可以通过 ZooKeeper 客户端查看所设置的参数，对应的 ZooKeeper 节点为/config/topics/[topic]，示例如下：

```
[zk: localhost:2181/kafka(CONNECTED) 7] get
/config/topics/topic-config
{"version":1,"config":
{"max.message.bytes":"10000","cleanup.policy":"co
mpact"}}
```

创建主题时对于主题名称的命名方式也很有讲究。首先是不能与已经存在的主题同名，如果创建了同名的主题就会报错。我们尝试创建一个已经存在的主题 topic-create，示例如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
create --topic topic-create --replication-factor
1 --partitions 1
Error while executing topic command : Topic
'topic-create' already exists.
[2018-09-08 23:04:29,542] ERROR
org.apache.kafka.common.errors.TopicExists-
Exception: Topic 'topic-create' already exists.
 (kafka.admin.TopicCommand$)
```

通过上面的示例可以看出，在发生命名冲突时会报出 TopicExistsException 的异常信息。在 kafka-topics.sh 脚本中还提供了一个 if-not-exists 参数，如果在创建主题时带上了这个参数，那么在发生命名冲突时将不做任何处理（既不创建主题，也不报错）。如果没有发生命名冲突，那么和不带 if-not-exists 参数的行为一样正常创建主题。我们再次尝试创建一个已经存在的主题 topic-create，示例如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
create --topic topic-create --replication-factor
1 --partitions 1 --if-not- exists

[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
describe --topic topic-create
Topic:topic-create  PartitionCount:4
ReplicationFactor:2 Configs:
        Topic: topic-create Partition: 0
Leader: 2   Replicas: 2,0   Isr: 2,0
        Topic: topic-create Partition: 1
Leader: 0   Replicas: 0,1   Isr: 0,1
        Topic: topic-create Partition: 2
Leader: 2   Replicas: 1,2   Isr: 2,1
        Topic: topic-create Partition: 3
Leader: 2   Replicas: 2,1   Isr: 2,1
```

通过上面的示例可以看出，在添加 if-not-exists 参数之后，并没有像第一次创建主题时的那样出现“Created topic "topic-create".”的提示信息。通过 describe 指令查看主题中的分区数和副本因子数，还是同第一次创建时的一样分别为4和2，也并没有被覆盖，如此便证实了 if-not-exists 参数可以在发生命名冲突时不做任何处理。在实际应用中，如果不想在创建主题的时候跳出 TopicExistsException 的异常信息，不妨试一下这个参数。

kafka-topics.sh 脚本在创建主题时还会检测是否包含“.”或“_”字符。为什么要检测这两个字符呢？因为在 Kafka 的内部做埋点时会根据主题的名称来命名 metrics 的名称，并且会将点号“.”改成下画线“_”。假设遇到一个名称为“topic.1_2”的主题，还有一个名称为“topic_1.2”的主题，那么最后的 metrics 的名称都会

为“topic_1_2”，这样就发生了名称冲突。举例如下，首先创建一个以“topic.1_2”为名称的主题，提示 WARNING 警告，之后再创建“topic_1.2”时发生 InvalidTopicException 异常。

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
create --replication-factor 1 --partitions 1 --
topic topic.1_2
WARNING: Due to limitations in metric names,
topics with a period ('.') or underscore ('_')
could collide. To avoid issues it is best to use
either, but not both.
Created topic "topic.1_2".

[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
create --replication-factor 1 --partitions 1 --
topic topic_1.2
WARNING: Due to limitations in metric names,
topics with a period ('.') or underscore ('_')
could collide. To avoid issues it is best to use
either, but not both.
Error while executing topic command : Topic
'topic_1.2' collides with existing topics:
topic.1_2
[2018-09-09 00:21:41,113] ERROR
org.apache.kafka.common.errors.InvalidTopicExcept
ion: Topic 'topic_1.2' collides with existing
topics: topic.1_2
 (kafka.admin.TopicCommand$)
```

注意要点：主题的命名同样不推荐（虽然可以这样做）使用双下画线“__”开头，因为以双下画线开头的主题一般看作 Kafka 的内部主题，比如__consumer_offsets 和 _transaction_state。*主题的名称必须由大小写字母、数字、点号“.”、连接线“-”、下画线“_”*组成，不能为空，不能只有点号“.”，也不能只有双点号“..”，且长度不能超过249。

Kafka 从0.10.x版本开始支持指定 broker 的机架信息（机架的名称）。如果指定了机架信息，则在分区副本分配时会尽可能地让分区副本分配到不同的机架上。指定机架信息是通过 broker 端参数 broker.rack 来配置的，比如配置当前 broker 所在的机架为“RACK1”：

```
broker.rack=RACK1
```

如果一个集群中有部分 broker 指定了机架信息，并且其余的 broker 没有指定机架信息，那么在执行 kafka-topics.sh 脚本创建主题时会报出的 AdminOperationException 的异常，示例如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
create --topic topic-create-rack -replication-
factor 1 --partitions 1
Error while executing topic command : Not all
brokers have rack information. Add --disable-
rack-aware in command line to make replica
assignment without rack information.
[2018-09-09 14:52:32,723] ERROR
kafka.admin.AdminOperationException: Not all
brokers have rack information. Add --disable-
rack-aware in command line to make replica
assignment without rack information.
…（省略若干）
```

此时若要成功创建主题，要么将集群中的所有 broker 都加上机架信息或都去掉机架信息，要么使用 disable-rack-aware 参数来忽略机架信息，示例如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
create --topic topic-create-rack -replication-
factor 1 --partitions 1 --disable-rack-aware
Created topic "topic-create-rack".
```

如果集群中的所有 broker 都有机架信息，那么也可以使用 disable-rack-aware 参数来忽略机架信息对分区副本的分配影响，有关分区副本的分配细节会在下一节中做详细介绍。

本节开头就提及了 kafka-topics.sh 脚本实质上是调用了 kafka.admin.TopicCommand 类，通过向 TopicCommand 类中传入一些关键参数来实现主题的管理。我们也可以直接调用

TopicCommand 类中的 main() 函数来直接管理主题，比如这里创建一个分区数为1、副本因子为1的主题 topic-create-api，如代码清单16-1所示。

```
//代码清单16-1 使用TopicCommand创建主题
public static void createTopic(){
    String[] options = new String[]{
            "--zookeeper",
"localhost:2181/kafka",
            "--create",
            "--replication-factor", "1",
            "--partitions", "1",
            "--topic", "topic-create-api"
    };
    kafka.admin.TopicCommand.main(options);
}
```

使用这种方式需要添加相应的 Maven 依赖：

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka_2.11</artifactId>
    <version>2.0.0</version>
</dependency>
```

可以看到这种方式与使用 kafka-topics.sh 脚本的方式并无太大差别，可以使用这种方式集成到自动化管理系统中来创建相应的主题。当然这种方式也可以适用于对主题的删、改、查等操作的实现，只需修改对应的参数即可。不过更推荐使用第20节中介绍的 KafkaAdminClient 来代替这种实现方式。