初识KafkaAdminClient

一般情况下，我们都习惯使用 kafka-topics.sh 脚本来管理主题，但有些时候我们希望将主题管理类的功能集成到公司内部的系统中，打造集管理、监控、运维、告警为一体的生态平台，那么就需要以程序调用 API 的方式去实现。本节主要介绍 KafkaAdminClient 的基本使用方式，以及采用这种调用 API 方式下的创建主题时的合法性验证。

基本使用

代码清单16-1中使用 TopicCommand 创建了一个主题，当然我们也可以用它来实现主题的删除、修改、查看等操作，实质上与使用 kafka-config.sh 脚本的方式无异。这种方式与应用程序之间的交互性非常差，且不说它的编程模型类似于拼写字符串，它本身调用的 TopicCommand 类的 main() 方法的返回值是一个 void 类，并不能提供给调用者有效的反馈信息。比如我们使用下面的方式来查看主题 topic-create 的详细信息，如代码清单20-1所示。

```
//代码清单20-1查看主题
public static void describeTopic(){
    String[] options = new String[]{
            "--zookeeper",
"localhost:2181/kafka",
            "--describe",
            "--topic", "topic-create"
    };
    kafka.admin.TopicCommand.main(options);
}
```

当调用 describeTopic() 方法时，虽然我们可以在终端看到主题 topic-create 的详细信息，但方法的调用者却无法捕获这个信息，因为返回值类型为 void。对于方法的调用者而言，执行这个方法和不执行这个方法没有什么区别。

在 Kafka 0.11.0.0 版本之前，我们可以通过 kafka-core 包（Kafka 服务端代码）下的 kafka.admin.AdminClient 和 kafka.admin.AdminUtils 来实现部分 Kafka 的管理功能，但它们都已经过时了，在未来的版本中会被删除。从0.11.0.0版本开始，Kafka 提供了另一个工具类 org.apache.kafka.clients.admin.KafkaAdminClient 来作为替代方案。KafkaAdminClient 不仅可以用来管理 broker、配置和 ACL（Access Control List），还可以用来管理主题。

KafkaAdminClient 继承了 org.apache.kafka.clients.admin.AdminClient 抽象类，并提供了多种方法。篇幅限制，下面只列出与本章内容相关的一些方法。

- 创建主题：CreateTopicsResult createTopics(Collection newTopics)。
- 删除主题：DeleteTopicsResult deleteTopics(Collection topics)。
- 列出所有可用的主题：ListTopicsResult listTopics()。
- 查看主题的信息：DescribeTopicsResult describeTopics(Collection topicNames)。
- 查询配置信息：DescribeConfigsResult describeConfigs(Collection resources)。
- 修改配置信息：AlterConfigsResult alterConfigs(Map<ConfigResource, Config> configs)。
- 增加分区：CreatePartitionsResult createPartitions(Map<String, NewPartitions> newPartitions)。

下面分别介绍这些方法的具体使用方式。首先分析如何使用KafkaAdminClient 创建一个主题，下面的示例中创建了一个分区数为4、副本因子为1的主题 topic-admin，如代码清单20-2所示。

```
//代码清单20-2 使用KafkaAdminClient创建一个主题
String brokerList =  "localhost:9092";
String topic = "topic-admin";

Properties props = new Properties();    ①
props.put(AdminClientConfig.BOOTSTRAP_SERVERS_CON
FIG, brokerList);
props.put(AdminClientConfig.REQUEST_TIMEOUT_MS_CO
NFIG, 30000);
AdminClient client = AdminClient.create(props);
②

NewTopic newTopic = new NewTopic(topic, 4,
(short) 1);  ③
CreateTopicsResult result = client.

createTopics(Collections.singleton(newTopic));  ④
try {
    result.all().get();
⑤
} catch (InterruptedException |
ExecutionException e) {
    e.printStackTrace();
}
client.close();
⑥
```

示例中第②行创建了一个 KafkaAdminClient 实例，实例中通过引入在第①行中建立的配置来连接 Kafka 集群。
AdminClient.create() 方法实际上调用的就是 KafkaAdminClient 中的 createInternal 方法构建的 KafkaAdminClient 实例，具体定义如下：

```
public static AdminClient create(Properties
props) {
    return KafkaAdminClient.createInternal(
            new AdminClientConfig(props), null);
}
```

第③行中的 NewTopic 用来设定所要创建主题的具体信息，包含创建主题时需要的主题名称、分区数和副本因子等。NewTopic 中的成员变量如下所示。

```
private final String name;  //主题名称
private final int numPartitions;    //分区数
private final short replicationFactor;  //副本因子
private final Map<Integer, List<Integer>>
replicasAssignments;  //分配方案
private Map<String, String> configs = null;
//配置
```

同 kafka-topics.sh 脚本一样，可以通过指定分区数和副本因子来创建一个主题，也可以通过指定区副本的具体分配方案来创建一个主题，比如将第③行替换为下面的内容：

```
Map<Integer, List<Integer>> replicasAssignments =
new HashMap<>();
replicasAssignments.put(0, Arrays.asList(0));
replicasAssignments.put(1, Arrays.asList(0));
replicasAssignments.put(2, Arrays.asList(0));
replicasAssignments.put(3, Arrays.asList(0));
NewTopic newTopic = new NewTopic(topic,
replicasAssignments);
```

也可以在创建主题时指定需要覆盖的配置。比如覆盖 cleanup.policy 配置，需要在第③和第④行之间加入如下代码：

```
Map<String, String> configs = new HashMap<>();
configs.put("cleanup.policy", "compact");
newTopic.configs(configs);
```

第④行是真正的创建主题的核心。KafkaAdminClient 内部使用 Kafka 的一套自定义二进制协议来实现诸如创建主题的管理功能。它主要的实现步骤如下：

1. 客户端根据方法的调用创建相应的协议请求，比如创建主题的 createTopics 方法，其内部就是发送 CreateTopicRequest 请求。
2. 客户端将请求发送至服务端。
3. 服务端处理相应的请求并返回响应，比如这个与 CreateTopicRequest 请求对应的就是 CreateTopicResponse。
4. 客户端接收相应的响应并进行解析处理。和协议相关的请求和相应的类基本都在 org.apache.kafka.common.requests 包下，AbstractRequest 和 AbstractResponse 是这些请求和响应类的两个基本父类。

有关 Kafka 的自定义协议的更多内容可以参阅[《图解Kafka之核心原理》(https://juejin.im/book/5c7d270ff265da2d89634e9e)](https://juejin.im/book/5c7d270ff265da2d89634e9e)的相关章节。

第④行中的返回值是 CreateTopicsResult 类型，它的具体定义也很简单，如代码清单20-3所示。

```
//代码清单20-3 CreateTopicsResult的具体内容
public class CreateTopicsResult {
    private final Map<String, KafkaFuture<Void>>
futures;

    CreateTopicsResult(Map<String,
KafkaFuture<Void>> futures) {
        this.futures = futures;
    }

    public Map<String, KafkaFuture<Void>>
values() {
        return futures;
    }

    public KafkaFuture<Void> all() {
        return KafkaFuture.allOf(futures.values()
                .toArray(new KafkaFuture[0]));
    }
}
```

CreateTopicsResult 中的方法主要还是针对成员变量 futures 的操作，futures 的类型 Map<String, KafkaFuture> 中的 key 代表主题名称，而 KafkaFuture 代表创建后的返回值类型。
KafkaAdminClient 中的 createTopics() 方法可以一次性创建多个主题。KafkaFuture 是原本为了支持JDK8以下的版本而自定义实现

的一个类，实现了 Future 接口，可以通过 Future.get() 方法来等待服务端的返回，参见代码清单20-2中的第⑤行。在未来的版本中，会有计划地将 KafkaFuture 替换为JDK8中引入的 CompletableFuture。

虽然这里创建主题之后的返回值类型为 Void，但并不代表所有操作的返回值类型都是 Void，比如 KafkaAdminClient 中的 listTopics() 方法的返回值为 ListTopicsResult 类型，这个 ListTopicsResult 类型内部的成员变量 future 的类型为 KafkaFuture<Map<String, TopicListing>>，这里就包含了具体的返回信息。

在使用 KafkaAdminClient 之后记得要调用 close() 方法来释放资源。

KafkaAdminClient 中的 deleteTopics()、listTopics() 及 describeTopics() 方法都很简单，读者不妨自己实践一下。下面讲一讲 describeConfigs() 和 alterConfigs() 这两个方法。首先查看刚刚创建的主题 topic-admin 的具体配置信息，如代码清单20-4所示。

```
//代码清单20-4 describeConfigs()方法的使用示例
public static void describeTopicConfig() throws
ExecutionException,
        InterruptedException {
    String brokerList =  "localhost:9092";
    String topic = "topic-admin";

    Properties props = new Properties();

props.put(AdminClientConfig.BOOTSTRAP_SERVERS_CON
FIG, brokerList);

props.put(AdminClientConfig.REQUEST_TIMEOUT_MS_CO
NFIG, 30000);
    AdminClient client =
AdminClient.create(props);

    ConfigResource resource =
            new
ConfigResource(ConfigResource.Type.TOPIC,
topic);①
    DescribeConfigsResult result =

client.describeConfigs(Collections.singleton(reso
urce));②
    Config config =
result.all().get().get(resource);③
    System.out.println(config);④
    client.close();
}
```

最终的输出结果不会只列出被覆盖的配置信息，而是会列出主题中所有的配置信息。
alterConfigs() 方法的使用方式也很简单。下面的示例中将主题 topic-admin 的 cleanup.policy 参数修改为 compact，只需将代码清单20-4中的第①至第④行替换为下面的内容即可：

```
ConfigResource resource = new
ConfigResource(ConfigResource.Type.TOPIC, topic);
ConfigEntry entry = new
ConfigEntry("cleanup.policy", "compact");
Config config = new
Config(Collections.singleton(entry));
Map<ConfigResource, Config> configs = new
HashMap<>();
configs.put(resource, config);
AlterConfigsResult result =
client.alterConfigs(configs);
result.all().get();
```

本章的最后将演示如何使用 KafkaAdminClient 的 createPartitions() 方法来增加一个主题的分区。下面的示例将主题 topic-admin 的分区从4增加到5，只需将代码清单20-4中的第①至第④行替换为下面的内容即可：

```
NewPartitions newPartitions =
NewPartitions.increaseTo(5);
Map<String, NewPartitions> newPartitionsMap = new
HashMap<>();
newPartitionsMap.put(topic, newPartitions);
CreatePartitionsResult result =
client.createPartitions(newPartitionsMap);
result.all().get();
```

主题合法性验证

一般情况下，Kafka 生产环境中的 auto.create.topics.enable 参数会被设置为 false，即自动创建主题这条路会被堵住。kafka-topics.sh 脚本创建的方式一般由运维人员操作，普通用户无权过问。那么 KafkaAdminClient 就为普通用户提供了一个“口子”，或者将其集成到公司内部的资源申请、审核系统中会更加方便。

普通用户在创建主题的时候，有可能由于误操作或其他原因而创建了不符合运维规范的主题，比如命名不规范，副本因子数太低等，这些都会影响后期的系统运维。如果创建主题的操作封装在资源申请、审核系统中，那么在前端就可以根据规则过滤不符合规范的申请操作。如果用户用 KafkaAdminClient 或类似的工具创建了一个错误的主题，我们有什么办法可以做相应的规范处理呢?

Kafka broker 端有一个这样的参数：create.topic.policy.class.name，默认值为 null，它提供了一个入口用来验证主题创建的合法性。使用方式很简单，只需要自定义实现 org.apache.kafka.server.policy.CreateTopicPolicy 接口，比如下面示例中的 PolicyDemo。然后在 broker 端的配置文件 config/server.properties 中配置参数 create.topic.policy.class.name 的值为 org.apache.kafka.server.policy.PolicyDemo，最后启动服务。

PolicyDemo 的代码参考代码清单20-5，主要实现接口中的 configure()、close() 及 validate() 方法，configure() 方法会在 Kafka 服务启动的时候执行，validate() 方法用来鉴定主题参数的合法性，其在创建主题时执行，close() 方法在关闭 Kafka 服务时执行。

```
//代码清单20-5 主题合法性验证示例
public class PolicyDemo implements
CreateTopicPolicy {
```

```
    public void configure(Map<String, ?> configs)
{
    }

    public void close() throws Exception {
    }

    public void validate(RequestMetadata
requestMetadata)
            throws PolicyViolationException {
        if (requestMetadata.numPartitions() !=
null ||

requestMetadata.replicationFactor() != null) {
            if (requestMetadata.numPartitions() <
5) {
                throw new
PolicyViolationException("Topic should have at "
+
                        "least 5 partitions,
received: "+

requestMetadata.numPartitions());
            }
            if
(requestMetadata.replicationFactor() <= 1) {
                throw new
PolicyViolationException("Topic should have at "
+
                        "least 2 replication
factor, recevied: "+

requestMetadata.replicationFactor());
```

```
            }
        }
    }
}
```

此时如果采用代码清单20-3中的方式创建一个分区数为4、副本因子为1的主题，那么客户端就出报出如下的错误：

```
java.util.concurrent.ExecutionException:
org.apache.kafka.common.errors.PolicyViolationExc
eption: Topic should have at least 5 partitions,
received: 4
```

相应的 Kafka 服务端的日志如下：

```
CreateTopicPolicy.RequestMetadata(topic=topic-
test2, numPartitions=4, replicationFactor=1,
replicasAssignments=null, configs={})
[2018-04-18 19:52:02,747] INFO [Admin Manager on
Broker 0]: Error processing create topic request
for topic topic-test2 with arguments
(numPartitions=4, replicationFactor=1,
replicasAssignments={}, configs={})
(kafka.server.AdminManager)
org.apache.kafka.common.errors.PolicyViolationExc
eption: Topic should have at least 5 partitions,
received: 4
```