消费者客户端开发

在了解了消费者与消费组之间的概念之后，我们就可以着手进行消费者客户端的开发了。在 Kafka 的历史中，消费者客户端同生产者客户端一样也经历了两个大版本：第一个是于 Kafka 开源之初使用 Scala 语言编写的客户端，我们可以称之为旧消费者客户端（Old Consumer）或 Scala 消费者客户端；第二个是从 Kafka 0.9.x 版本开始推出的使用 Java 编写的客户端，我们可以称之为新消费者客户端（New Consumer）或 Java 消费者客户端，它弥补了旧客户端中存在的诸多设计缺陷

本节主要介绍目前流行的新消费者（Java 语言编写的）客户端，而旧消费者客户端已被淘汰，故不再做相应的介绍了。

一个正常的消费逻辑需要具备以下几个步骤：

1. 配置消费者客户端参数及创建相应的消费者实例。
2. 订阅主题。
3. 拉取消息并消费。
4. 提交消费位移。
5. 关闭消费者实例。

代码清单2-2中已经简单对消费者客户端的编码做了演示，本节对其稍做修改，如代码清单8-1所示。

```
//代码清单8-1 消费者客户端示例
public class KafkaConsumerAnalysis {
    public static final String brokerList =
"localhost:9092";
    public static final String topic = "topic-
demo";
    public static final String groupId =
```

```
"group.demo";
    public static final AtomicBoolean isRunning =
new AtomicBoolean(true);

    public static Properties initConfig(){
        Properties props = new Properties();
        props.put("key.deserializer",

"org.apache.kafka.common.serialization.StringDese
rializer");
        props.put("value.deserializer",

"org.apache.kafka.common.serialization.StringDese
rializer");
        props.put("bootstrap.servers",
brokerList);
        props.put("group.id", groupId);
        props.put("client.id",
"consumer.client.id.demo");
        return props;
    }

    public static void main(String[] args) {
        Properties props = initConfig();
        KafkaConsumer<String, String> consumer =
new KafkaConsumer<>(props);
        consumer.subscribe(Arrays.asList(topic));

        try {
            while (isRunning.get()) {
                ConsumerRecords<String, String>
records =
```

```
consumer.poll(Duration.ofMillis(1000));
                for (ConsumerRecord<String,
String> record : records) {
                    System.out.println("topic = "
+ record.topic()
                            + ", partition = "+
record.partition()
                            + ", offset = " +
record.offset());
                    System.out.println("key = " +
record.key()
                            + ", value = " +
record.value());
                    //do something to process
record.
                }
            }
        } catch (Exception e) {
            log.error("occur exception ", e);
        } finally {
            consumer.close();
        }
    }
}
```

相比于代码清单2-2而言，修改过后的代码多了一点东西，我们按照消费逻辑的各个步骤来做相应的分析。

必要的参数配置

在创建真正的消费者实例之前需要做相应的参数配置，比如上一节中的设置消费者所属的消费组的名称、连接地址等。参照代码清单8-1中的 initConfig() 方法，在 Kafka 消费者客户端 KafkaConsumer

中有4个参数是必填的。

- bootstrap.servers：该参数的释义和生产者客户端 KafkaProducer 中的相同，用来指定连接 Kafka 集群所需的 broker 地址清单，具体内容形式为 host1:port1,host2:post，可以设置一个或多个地址，中间用逗号隔开，此参数的默认值为“”。注意这里并非需要设置集群中全部的 broker 地址，消费者会从现有的配置中查找到全部的 Kafka 集群成员。这里设置两个以上的 broker 地址信息，当其中任意一个宕机时，消费者仍然可以连接到 Kafka 集群上。
- group.id：消费者隶属的消费组的名称，默认值为“”。如果设置为空，则会报出异常：Exception in thread "main" org.apache.kafka.common.errors.InvalidGroupIdExceptio The configured groupId is invalid。一般而言，这个参数需要设置成具有一定的业务意义的名称。
- key.deserializer 和 value.deserializer：与生产者客户端 KafkaProducer 中的 key.serializer和value.serializer 参数对应。消费者从 broker 端获取的消息格式都是字节数组（byte[]）类型，所以需要执行相应的反序列化操作才能还原成原有的对象格式。这两个参数分别用来指定消息中 key 和 value 所需反序列化操作的反序列化器，这两个参数无默认值。注意这里必须填写反序列化器类的全限定名，比如示例中的 org.apache.kafka.common.serialization.StringDeserialize 单单指定 StringDeserializer 是错误的。有关更多的反序列化内容可以参考下一节。

注意到代码清单8-1中的 initConfig() 方法里还设置了一个参数 client.id，这个参数用来设定 KafkaConsumer 对应的客户端id，默认值也为“”。如果客户端不设置，则 KafkaConsumer 会自动生成一个非空字符串，内容形式如“consumer-1”、“consumer-2”，即字符串“consumer-”与数字的拼接。

KafkaConsumer 中的参数众多，远非示例 initConfig() 方法中的那样只有5个，开发人员可以根据业务应用的实际需求来修改这些参数的默认值，以达到灵活调配的目的。一般情况下，普通开发人员无法全部记住所有的参数名称，只能有个大致的印象，在实际使用过程中，诸如“key.deserializer”、“auto.offset.reset”之类的字符串经常由于人为因素而书写错误。为此，我们可以直接使用客户端中的 org.apache.kafka.clients.consumer.ConsumerConfig 类来做一定程度上的预防，每个参数在 ConsumerConfig 类中都有对应的名称，就以代码清单8-1中的 initConfig() 方法为例，引入 ConsumerConfig 后的修改结果如下：

```
    public static Properties initConfig(){
        Properties props = new Properties();

props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_C
ONFIG,

"org.apache.kafka.common.serialization.StringDese
rializer");

props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS
_CONFIG,

"org.apache.kafka.common.serialization.StringDese
rializer");

props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG
, brokerList);
        props.put(ConsumerConfig.GROUP_ID_CONFIG,
groupId);

props.put(ConsumerConfig.CLIENT_ID_CONFIG,
"client.id.demo");
        return props;
    }
```

注意到上面的代码中 key.deserializer 和 value.deserializer 参数对应类的全限定名比较长，也比较容易写错，这里通过 Java 中的技巧来做进一步的改进，相关代码如下：

```
props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_C
ONFIG,

StringDeserializer.class.getName());

props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS
_CONFIG,

StringDeserializer.class.getName());
```

如此代码就简洁了许多，同时也预防了人为出错的可能。在配置完参数之后，我们就可以使用它来创建一个消费者实例：

```
KafkaConsumer<String, String> consumer = new
KafkaConsumer<>(props);
```

本节介绍的 KafkaConsumer 配置相关的内容基本上和介绍 KafkaProducer 配置时的一样，除了配置对应的反序列化器，只多了一个必要的 group.id 参数。

订阅主题和分区

在创建好消费者之后，我们就需要为该消费者订阅相关的主题了。一个消费者可以订阅一个或多个主题，代码清单8-1中我们使用 subscribe() 方法订阅了一个主题，对于这个方法而言，既可以以集合的形式订阅多个主题，也可以以正则表达式的形式订阅特定模式的主题。subscribe 的几个重载方法如下：

```
public void subscribe(Collection<String> topics,
    ConsumerRebalanceListener listener)
public void subscribe(Collection<String> topics)
public void subscribe(Pattern pattern,
ConsumerRebalanceListener listener)
public void subscribe(Pattern pattern)
```

对于消费者使用集合的方式（subscribe(Collection)）来订阅主题而言，比较容易理解，订阅了什么主题就消费什么主题中的消息。如果前后两次订阅了不同的主题，那么消费者以最后一次的为准。

```
consumer.subscribe(Arrays.asList(topic1));
consumer.subscribe(Arrays.asList(topic2));
```

上面的示例中，最终消费者订阅的是 topic2，而不是 topic1，也不是 topic1 和 topic2 的并集。

如果消费者采用的是正则表达式的方式（subscribe(Pattern)）订阅，在之后的过程中，如果有人又创建了新的主题，并且主题的名字与正则表达式相匹配，那么这个消费者就可以消费到新添加的主题中的消息。如果应用程序需要消费多个主题，并且可以处理不同的类型，那么这种订阅方式就很有效。在 Kafka 和其他系统之间进行数据复制时，这种正则表达式的方式就显得很常见。正则表达式的方式订阅的示例如下：

```
consumer.subscribe(Pattern.compile("topic-.*"));
```

细心的读者可能观察到在 subscribe 的重载方法中有一个参数类型是 ConsumerRebalance- Listener，这个是用来设置相应的再均衡监听器的，具体的内容可以参考第13节的相关内容。

消费者不仅可以通过 KafkaConsumer.subscribe() 方法订阅主题，还可以直接订阅某些主题的特定分区，在 KafkaConsumer 中还提供了一个 assign() 方法来实现这些功能，此方法的具体定义如下：

```
public void assign(Collection<TopicPartition>
partitions)
```

这个方法只接受一个参数 partitions，用来指定需要订阅的分区集合。这里补充说明一下 TopicPartition 类，在 Kafka 的客户端中，它用来表示分区，这个类的部分内容如下所示。

```
public final class TopicPartition implements
Serializable {

    private final int partition;
    private final String topic;

    public TopicPartition(String topic, int
partition) {
        this.partition = partition;
        this.topic = topic;
    }

    public int partition() {
        return partition;
    }

    public String topic() {
        return topic;
    }
    //省略hashCode()、equals()和toString()方法
}
```

TopicPartition 类只有2个属性：topic 和 partition，分别代表分区所属的主题和自身的分区编号，这个类可以和我们通常所说的主题—分区的概念映射起来。

我们将代码清单8-1中的 subscribe() 方法修改为 assign() 方法，

这里只订阅 topic-demo 主题中分区编号为0的分区，相关代码如下：

```
consumer.assign(Arrays.asList(new
TopicPartition("topic-demo", 0)));
```

有读者会有疑问：如果我们事先并不知道主题中有多少个分区怎么办？KafkaConsumer 中的 partitionsFor() 方法可以用来查询指定主题的元数据信息，partitionsFor() 方法的具体定义如下：

```
public List<PartitionInfo> partitionsFor(String
topic)
```

其中 PartitionInfo 类型即为主题的分区元数据信息，此类的主要结构如下：

```
public class PartitionInfo {
    private final String topic;
    private final int partition;
    private final Node leader;
    private final Node[] replicas;
    private final Node[] inSyncReplicas;
    private final Node[] offlineReplicas;
        //这里省略了构造函数、属性提取、toString等方法
}
```

PartitionInfo 类中的属性 topic 表示主题名称，partition 代表分区编号，leader 代表分区的 leader 副本所在的位置，replicas 代表分区的 AR 集合，inSyncReplicas 代表分区的 ISR 集合，offlineReplicas 代表分区的 OSR 集合。
通过 partitionFor() 方法的协助，我们可以通过 assign() 方法来实现订阅主题（全部分区）的功能，示例参考如下：

```
List<TopicPartition> partitions = new ArrayList<>
();
List<PartitionInfo> partitionInfos =
consumer.partitionsFor(topic);
if (partitionInfos != null) {
    for (PartitionInfo tpInfo : partitionInfos) {
        partitions.add(new
TopicPartition(tpInfo.topic(),
tpInfo.partition()));
    }
}
consumer.assign(partitions);
```

既然有订阅，那么就有取消订阅，可以使用 KafkaConsumer 中的 unsubscribe() 方法来取消主题的订阅。这个方法既可以取消通过 subscribe(Collection) 方式实现的订阅，也可以取消通过 subscribe(Pattern) 方式实现的订阅，还可以取消通过 assign(Collection) 方式实现的订阅。示例代码如下：

```
consumer.unsubscribe();
```

如果将 subscribe(Collection) 或 assign(Collection) 中的集合参数设置为空集合，那么作用等同于 unsubscribe() 方法，下面示例中的三行代码的效果相同：

```
consumer.unsubscribe();
consumer.subscribe(new ArrayList<String>());
consumer.assign(new ArrayList<TopicPartition>());
```

如果没有订阅任何主题或分区，那么再继续执行消费程序的时候会报出 IllegalStateException 异常：

```
java.lang.IllegalStateException: Consumer is not
subscribed to any topics or assigned any
partitions
```

集合订阅的方式 subscribe(Collection)、正则表达式订阅的方式 subscribe(Pattern) 和指定分区的订阅方式 assign(Collection) 分表代表了三种不同的订阅状态：AUTO_TOPICS、AUTO_PATTERN 和 USER_ASSIGNED（如果没有订阅，那么订阅状态为 NONE）。然而这三种状态是互斥的，在一个消费者中只能使用其中的一种，否则会报出 IllegalStateException 异常：

```
java.lang.IllegalStateException: Subscription to
topics, partitions and pattern are mutually
exclusive.
```

通过 subscribe() 方法订阅主题具有消费者自动再均衡的功能，在多个消费者的情况下可以根据分区分配策略来自动分配各个消费者与分区的关系。当消费组内的消费者增加或减少时，分区分配关系会自动调整，以实现消费负载均衡及故障自动转移。而通过 assign() 方法订阅分区时，是不具备消费者自动均衡的功能的，其实这一点从 assign() 方法的参数中就可以看出端倪，两种类型的 subscribe() 都有 ConsumerRebalanceListener 类型参数的方法，而 assign() 方法却没有。