消息消费

Kafka 中的消费是基于拉模式的。消息的消费一般有两种模式：推模式和拉模式。推模式是服务端主动将消息推送给消费者，而拉模式是消费者主动向服务端发起请求来拉取消息。

从代码清单8-1中可以看出，Kafka 中的消息消费是一个不断轮询的过程，消费者所要做的就是重复地调用 poll() 方法，而 poll() 方法返回的是所订阅的主题（分区）上的一组消息。

对于 poll() 方法而言，如果某些分区中没有可供消费的消息，那么此分区对应的消息拉取的结果就为空；如果订阅的所有分区中都没有可供消费的消息，那么 poll() 方法返回为空的消息集合。

poll() 方法的具体定义如下：

```
public ConsumerRecords<K, V> poll(final Duration
timeout)
```

注意到 poll() 方法里还有一个超时时间参数 timeout，用来控制 poll() 方法的阻塞时间，在消费者的缓冲区里没有可用数据时会发生阻塞。注意这里 timeout 的类型是 Duration，它是 JDK8 中新增的一个与时间有关的类型。在 Kafka 2.0.0 之前的版本中，timeout 参数的类型为 long，与此类型对应的 poll() 方法的具体定义如下：

```
@Deprecated
public ConsumerRecords<K, V> poll(final long
timeout)
```

poll(long) 方法中 timeout 的时间单位固定为毫秒，而 poll(Duration) 方法可以根据 Duration 中的 ofMillis()、ofSeconds()、ofMinutes()、ofHours() 等多种不同的方法指定不同

的时间单位，灵活性更强。并且 poll(long) 方法也已经被标注为 @Deprecated，虽然目前还可以使用，如果条件允许的话，还是推荐使用 poll(Duration) 的方式。

timeout 的设置取决于应用程序对响应速度的要求，比如需要在多长时间内将控制权移交给执行轮询的应用线程。可以直接将 timeout 设置为0，这样 poll() 方法会立刻返回，而不管是否已经拉取到了消息。如果应用线程唯一的工作就是从 Kafka 中拉取并消费消息，则可以将这个参数设置为最大值 Long.MAX_VALUE。

消费者消费到的每条消息的类型为 ConsumerRecord（注意与 ConsumerRecords 的区别），这个和生产者发送的消息类型 ProducerRecord 相对应，不过 ConsumerRecord 中的内容更加丰富，具体的结构参考如下代码：

```
public class ConsumerRecord<K, V> {
    private final String topic;
    private final int partition;
    private final long offset;
    private final long timestamp;
    private final TimestampType timestampType;
    private final int serializedKeySize;
    private final int serializedValueSize;
    private final Headers headers;
    private final K key;
    private final V value;
    private volatile Long checksum;
        //省略若干方法
}
```

topic 和 partition 这两个字段分别代表消息所属主题的名称和所在分区的编号。offset 表示消息在所属分区的偏移量。timestamp 表示时间戳，与此对应的 timestampType 表示时间戳的类型。

timestampType 有两种类型：CreateTime和LogAppendTime，分别代表消息创建的时间戳和消息追加到日志的时间戳。headers 表示消息的头部内容。

key 和 value 分别表示消息的键和消息的值，一般业务应用要读取的就是 value，比如使用第4节中的 CompanySerializer 序列化了一个 Company 对象，然后将其存入 Kafka，那么消费到的消息中的 value 就是经过 CompanyDeserializer 反序列化后的 Company 对象。serializedKeySize 和 serializedValueSize 分别表示 key 和 value 经过序列化之后的大小，如果 key 为空，则 serializedKeySize 值为-1。同样，如果 value 为空，则 serializedValueSize 的值也会为-1。checksum 是 CRC32 的校验值。

我们在消费消息的时候可以直接对 ConsumerRecord 中感兴趣的字段进行具体的业务逻辑处理。

poll() 方法的返回值类型是 ConsumerRecords，它用来表示一次拉取操作所获得的消息集，内部包含了若干 ConsumerRecord，它提供了一个 iterator() 方法来循环遍历消息集内部的消息，iterator() 方法的定义如下：

```
public Iterator<ConsumerRecord<K, V>> iterator()
```

在代码清单8-1中，我们使用这种方法来获取消息集中的每一个 ConsumerRecord。除此之外，我们还可以按照分区维度来进行消费，这一点很有用，在手动提交位移时尤为明显，有关位移提交的内容我们会在下一节中详细陈述。ConsumerRecords 类提供了一个 records(TopicPartition) 方法来获取消息集中指定分区的消息，此方法的定义如下：

```
public List<ConsumerRecord<K, V>>
records(TopicPartition partition)
```

我们不妨使用这个 records(TopicPartition) 方法来修改一下代码清单8-1中的消费逻辑，主要的示例代码如下：

```
ConsumerRecords<String, String> records =
         consumer.poll(Duration.ofMillis(1000));
for (TopicPartition tp : records.partitions()) {
    for (ConsumerRecord<String, String> record :
records.records(tp)) {
        System.out.println(record.partition()+" :
"+record.value());
    }
}
```

上面示例中的 ConsumerRecords.partitions() 方法用来获取消息集中所有分区。在 ConsumerRecords 类中还提供了按照主题维度来进行消费的方法，这个方法是 records(TopicPartition) 的重载方法，具体定义如下：

```
public Iterable<ConsumerRecord<K, V>>
records(String topic)
```

ConsumerRecords 类中并没提供与 partitions() 类似的 topics() 方法来查看拉取的消息集中所包含的主题列表，如果要按照主题维度来进行消费，那么只能根据消费者订阅主题时的列表来进行逻辑处理了。下面的示例演示了如何使用 ConsumerRecords 中的 record(String topic) 方法：

```
List<String> topicList = Arrays.asList(topic1,
topic2);
consumer.subscribe(topicList);
try {
    while (isRunning.get()) {
        ConsumerRecords<String, String> records =

consumer.poll(Duration.ofMillis(1000));
        for (String topic : topicList) {
            for (ConsumerRecord<String, String>
record :
                    records.records(topic)) {
                System.out.println(record.topic()
+ " : " + record.value());
            }
        }
    }
}finally {
    consumer.close();
}
```

在 ConsumerRecords 类中还提供了几个方法来方便开发人员对消息集进行处理：count() 方法用来计算出消息集中的消息个数，返回类型是 int；isEmpty() 方法用来判断消息集是否为空，返回类型是 boolean；empty() 方法用来获取一个空的消息集，返回类型是 ConsumerRecord<K，V>。

到目前为止，可以简单地认为 poll() 方法只是拉取一下消息而已，但就其内部逻辑而言并不简单，它涉及消费位移、消费者协调器、组协调器、消费者的选举、分区分配的分发、再均衡的逻辑、心跳等内容，在后面的章节中会循序渐进地介绍这些内容。