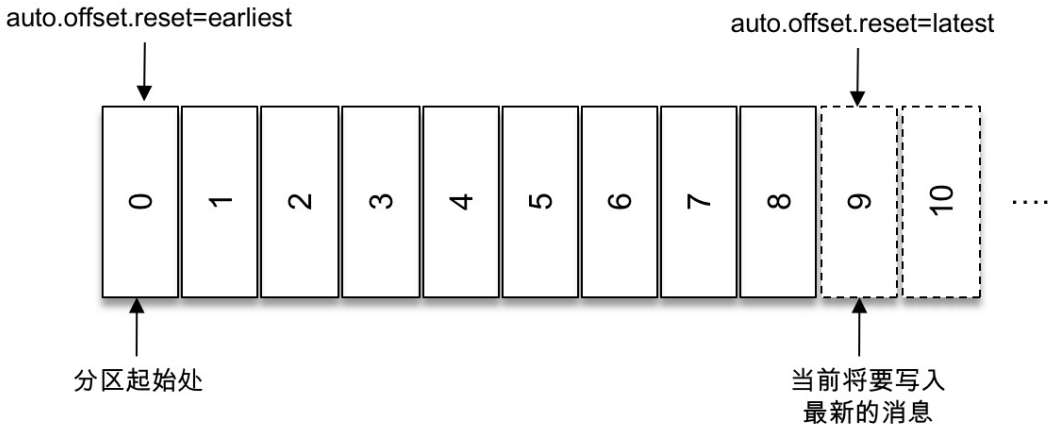
指定位移消费

在上一节中我们讲述了如何进行消费位移的提交，正是有了消费位移的持久化，才使消费者在关闭、崩溃或者在遇到再均衡的时候，可以让接替的消费者能够根据存储的消费位移继续进行消费。

试想一下，当一个新的消费组建立的时候，它根本没有可以查找的消费位移。或者消费组内的一个新消费者订阅了一个新的主题，它也没有可以查找的消费位移。当 __consumer_offsets 主题中有关这个消费组的位移信息过期而被删除后，它也没有可以查找的消费位移。



在 Kafka 中每当消费者查找不到所记录的消费位移时，就会根据消费者客户端参数 auto.offset.reset 的配置来决定从何处开始进行消费，这个参数的默认值为“latest”，表示从分区末尾开始消费消息。参考上图，按照默认的配置，消费者会从9开始进行消费（9是下一条要写入消息的位置），更加确切地说是从9开始拉取消息。如果将 auto.offset.reset 参数配置为“earliest”，那么消费者会从起始处，也就是0开始消费。

举个例子，在 auto.offset.reset 参数默认的配置下，用一个新的消费组来消费主题 topic-demo 时，客户端会报出重置位移的提示信息，参考如下：

```
[2018-08-18 18:13:16,029] INFO [Consumer
clientId=consumer-1, groupId=group.demo]
Resetting offset for partition topic-demo-3 to
offset 100.
[2018-08-18 18:13:16,030] INFO [Consumer
clientId=consumer-1, groupId=group.demo]
Resetting offset for partition topic-demo-0 to
offset 100.
[2018-08-18 18:13:16,030] INFO [Consumer
clientId=consumer-1, groupId=group.demo]
Resetting offset for partition topic-demo-2 to
offset 100.
[2018-08-18 18:13:16,031] INFO [Consumer
clientId=consumer-1, groupId=group.demo]
Resetting offset for partition topic-demo-1 to
offset 100.
```

除了查找不到消费位移，位移越界也会触发 auto.offset.reset 参数的执行，这个在下面要讲述的 seek 系列的方法中会有相关的介绍。

auto.offset.reset 参数还有一个可配置的值—“none”，配置为此值就意味着出现查到不到消费位移的时候，既不从最新的消息位置处开始消费，也不从最早的消息位置处开始消费，此时会报出 NoOffsetForPartitionException 异常，示例如下：

```
org.apache.kafka.clients.consumer.NoOffsetForPart
itionException: Undefined offset with no reset
policy for partitions: [topic-demo-3, topic-demo-
0, topic-demo-2, topic-demo-1].
```

如果能够找到消费位移，那么配置为“none”不会出现任何异常。如果配置的不是“latest”、“earliest”和“none”，则会报出 ConfigException 异常，示例如下：

```
org.apache.kafka.common.config.ConfigException:
Invalid value any for configuration
auto.offset.reset: String must be one of: latest,
earliest, none.
```

到目前为止，我们知道消息的拉取是根据 poll() 方法中的逻辑来处理的，这个 poll() 方法中的逻辑对于普通的开发人员而言是一个黑盒，无法精确地掌控其消费的起始位置。提供的 auto.offset.reset 参数也只能在找不到消费位移或位移越界的情况下粗粒度地从开头或末尾开始消费。有些时候，我们需要一种更细粒度的掌控，可以让我们从特定的位移处开始拉取消息，而 KafkaConsumer 中的 seek() 方法正好提供了这个功能，让我们得以追前消费或回溯消费。seek() 方法的具体定义如下：

```
public void seek(TopicPartition partition, long
offset)
```

seek() 方法中的参数 partition 表示分区，而 offset 参数用来指定从分区的哪个位置开始消费。seek() 方法只能重置消费者分配到的分区的消费位置，而分区的分配是在 poll() 方法的调用过程中实现的。也就是说，在执行 seek() 方法之前需要先执行一次 poll() 方法，等到分配到分区之后才可以重置消费位置。seek() 方法的使用示例如代码清单12-1所示（只列出关键代码）。

```
//代码清单12-1 seek方法的使用示例
KafkaConsumer<String, String> consumer = new
KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList(topic));
consumer.poll(Duration.ofMillis(10000));
①
Set<TopicPartition> assignment =
consumer.assignment();     ②
for (TopicPartition tp : assignment) {
    consumer.seek(tp, 10);
③
}
while (true) {
    ConsumerRecords<String, String> records =

consumer.poll(Duration.ofMillis(1000));
    //consume the record.
}
```

上面示例中第③行设置了每个分区的消费位置为10。第②行中的 assignment() 方法是用来获取消费者所分配到的分区信息的，这个方法的具体定义如下：

```
public Set<TopicPartition> assignment()
```

如果我们将代码清单12-1中第①行 poll() 方法的参数设置为0，即这一行替换为：

```
consumer.poll(Duration.ofMillis(0));
```

在此之后，会发现 seek() 方法并未有任何作用。因为当 poll() 方法中的参数为0时，此方法立刻返回，那么 poll() 方法内部进行分区分配的逻辑就会来不及实施。也就是说，消费者此时并未分配到任何分区，如此第②行中的 assignment 便是一个空列表，第③行代码也

不会执行。那么这里的 timeout 参数设置为多少合适呢？太短会使分配分区的动作失败，太长又有可能造成一些不必要的等待。我们可以通过 KafkaConsumer 的 assignment() 方法来判定是否分配到了相应的分区，参考下面的代码清单12-2：

```
//代码清单12-2 seek()方法的另一种使用示例
KafkaConsumer<String, String> consumer = new
KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList(topic));
Set<TopicPartition> assignment = new HashSet<>();
while (assignment.size() == 0) {//如果不为0，则说明
已经成功分配到了分区
    consumer.poll(Duration.ofMillis(100));
    assignment = consumer.assignment();
}
for (TopicPartition tp : assignment) {
    consumer.seek(tp, 10);
}
while (true) {
    ConsumerRecords<String, String> records =

consumer.poll(Duration.ofMillis(1000));
    //consume the record.
}
```

如果对未分配到的分区执行 seek() 方法，那么会报出 IllegalStateException 的异常。类似在调用 subscribe() 方法之后直接调用 seek() 方法：

```
consumer.subscribe(Arrays.asList(topic));
consumer.seek(new TopicPartition(topic,0),10);
```

会报出如下的异常：

```
java.lang.IllegalStateException: No current
assignment for partition topic-demo-0
```

如果消费组内的消费者在启动的时候能够找到消费位移，除非发生位移越界，否则 auto.offset.reset 参数并不会奏效，此时如果想指定从开头或末尾开始消费，就需要 seek() 方法的帮助了，代码清单12-3用来指定从分区末尾开始消费。

```
//代码清单12-3 使用seek()方法从分区末尾消费
KafkaConsumer<String, String> consumer = new
KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList(topic));
Set<TopicPartition> assignment = new HashSet<>();
while (assignment.size() == 0) {
    consumer.poll(Duration.ofMillis(100));
    assignment = consumer.assignment();
}
Map<TopicPartition, Long> offsets =
consumer.endOffsets(assignment);    ①
for (TopicPartition tp : assignment) {
    consumer.seek(tp, offsets.get(tp));
②
}
```

代码清单12-3中第①行的 endOffsets() 方法用来获取指定分区的末尾的消息位置，参考上图中9的位置，注意这里获取的不是8，是将要写入最新消息的位置。endOffsets 的具体方法定义如下：

```
public Map<TopicPartition, Long> endOffsets(
            Collection<TopicPartition>
partitions)
public Map<TopicPartition, Long> endOffsets(
            Collection<TopicPartition>
partitions,
            Duration timeout)
```

其中 partitions 参数表示分区集合，而 timeout 参数用来设置等待获取的超时时间。如果没有指定 timeout 参数的值，那么 endOffsets() 方法的等待时间由客户端参数 request.timeout.ms 来设置，默认值为30000。与 endOffsets 对应的是 beginningOffsets() 方法，一个分区的起始位置起初是0，但并不代表每时每刻都为0，因为日志清理的动作会清理旧的数据，所以分区的起始位置会自然而然地增加。beginningOffsets() 方法的具体定义如下：

```
public Map<TopicPartition, Long>
beginningOffsets(
            Collection<TopicPartition>
partitions)
public Map<TopicPartition, Long>
beginningOffsets(
            Collection<TopicPartition>
partitions,
            Duration timeout)
```

beginningOffsets() 方法中的参数内容和含义都与 endOffsets() 方法中的一样，配合这两个方法我们就可以从分区的开头或末尾开始消费。其实 KafkaConsumer 中直接提供了 seekToBeginning() 方法和 seekToEnd() 方法来实现这两个功能，这两个方法的具体定义如下：

```
public void
seekToBeginning(Collection<TopicPartition>
partitions)
public void seekToEnd(Collection<TopicPartition>
partitions)
```

有时候我们并不知道特定的消费位置，却知道一个相关的时间点，比如我们想要消费昨天8点之后的消息，这个需求更符合正常的思维逻辑。此时我们无法直接使用 seek() 方法来追溯到相应的位置。KafkaConsumer 同样考虑到了这种情况，它提供了一个 offsetsForTimes() 方法，通过 timestamp 来查询与此对应的分区位置。

```
public Map<TopicPartition, OffsetAndTimestamp>
offsetsForTimes(
            Map<TopicPartition, Long>
timestampsToSearch)
public Map<TopicPartition, OffsetAndTimestamp>
offsetsForTimes(
            Map<TopicPartition, Long>
timestampsToSearch,
            Duration timeout)
```

offsetsForTimes() 方法的参数 timestampsToSearch 是一个 Map 类型，key 为待查询的分区，而 value 为待查询的时间戳，该方法会返回时间戳大于等于待查询时间的第一条消息对应的位置和时间戳，对应于 OffsetAndTimestamp 中的 offset 和 timestamp 字段。

下面的示例演示了 offsetsForTimes() 和 seek() 之间的使用方法，首先通过 offsetForTimes() 方法获取一天之前的消息位置，然后使用 seek() 方法追溯到相应位置开始消费，示例中的 assignment 变量和代码清单12-3中的一样，表示消费者分配到的分区集合。

```
Map<TopicPartition, Long> timestampToSearch = new
HashMap<>();
for (TopicPartition tp : assignment) {
    timestampToSearch.put(tp,
System.currentTimeMillis()-1*24*3600*1000);
}
Map<TopicPartition, OffsetAndTimestamp> offsets =

consumer.offsetsForTimes(timestampToSearch);
for (TopicPartition tp : assignment) {
    OffsetAndTimestamp offsetAndTimestamp =
offsets.get(tp);
    if (offsetAndTimestamp != null) {
        consumer.seek(tp,
offsetAndTimestamp.offset());
    }
}
```

前面说过位移越界也会触发 auto.offset.reset 参数的执行，位移越界是指知道消费位置却无法在实际的分区中查找到，比如想要从上图中的位置10处拉取消息时就会发生位移越界。注意拉取上图中位置9处的消息时并未越界，这个位置代表特定的含义（LEO）。我们通过 seek() 方法来演示发生位移越界时的情形，将代码清单12-3中的第②行代码修改为：

```
consumer.seek(tp, offsets.get(tp)+1);
```

此时客户端会报出如下的提示信息：

```
[2018-08-19 16:13:44,700] INFO [Consumer
clientId=consumer-1, groupId=group.demo] Fetch
offset 101 is out of range for partition topic-
demo-3, resetting offset
[2018-08-19 16:13:44,701] INFO [Consumer
clientId=consumer-1, groupId=group.demo] Fetch
offset 101 is out of range for partition topic-
demo-0, resetting offset
[2018-08-19 16:13:44,701] INFO [Consumer
clientId=consumer-1, groupId=group.demo] Fetch
offset 101 is out of range for partition topic-
demo-2, resetting offset
[2018-08-19 16:13:44,701] INFO [Consumer
clientId=consumer-1, groupId=group.demo] Fetch
offset 101 is out of range for partition topic-
demo-1, resetting offset
[2018-08-19 16:13:44,708] INFO [Consumer
clientId=consumer-1, groupId=group.demo]
Resetting offset for partition topic-demo-3 to
offset 100.
[2018-08-19 16:13:44,708] INFO [Consumer
clientId=consumer-1, groupId=group.demo]
Resetting offset for partition topic-demo-0 to
offset 100.
[2018-08-19 16:13:44,709] INFO [Consumer
clientId=consumer-1, groupId=group.demo]
Resetting offset for partition topic-demo-2 to
offset 100.
[2018-08-19 16:13:44,713] INFO [Consumer
clientId=consumer-1, groupId=group.demo]
Resetting offset for partition topic-demo-1 to
offset 100.
```

通过上面加粗的提示信息可以了解到，原本拉取位置为101（fetch offset 101），但已经越界了（out of range），所以此时会根据 auto.offset.reset 参数的默认值来将拉取位置重置（resetting offset）为100，我们也能知道此时分区 topic-demo-3 中最大的消息 offset为99。

上一节中提及了 Kafka 中的消费位移是存储在一个内部主题中的，而本节的 seek() 方法可以突破这一限制：消费位移可以保存在任意的存储介质中，例如数据库、文件系统等。以数据库为例，我们将消费位移保存在其中的一个表中，在下次消费的时候可以读取存储在数据表中的消费位移并通过 seek() 方法指向这个具体的位置，伪代码如代码清单12-4所示。

```
//代码清单12-4 消费位移保存在DB中
consumer.subscribe(Arrays.asList(topic));
//省略poll()方法及assignment的逻辑
for(TopicPartition tp: assignment){
    long offset = getOffsetFromDB(tp);//从DB中读取
消费位移
    consumer.seek(tp, offset);
}
while(true){
    ConsumerRecords<String, String> records =

consumer.poll(Duration.ofMillis(1000));
    for (TopicPartition partition :
records.partitions()) {
        List<ConsumerRecord<String, String>>
partitionRecords =
                records.records(partition);
        for (ConsumerRecord<String, String>
record : partitionRecords) {
            //process the record.
        }
        long lastConsumedOffset =
partitionRecords
                .get(partitionRecords.size() -
1).offset();
         //将消费位移存储在DB中
        storeOffsetToDB(partition,
lastConsumedOffset+1);
    }
}
```

seek() 方法为我们提供了从特定位置读取消息的能力，我们可以通过这个方法来向前跳过若干消息，也可以通过这个方法来向后回溯若干消息，这样为消息的消费提供了很大的灵活性。seek() 方法也为我们提供了将消费位移保存在外部存储介质中的能力，还可以配合再均衡监听器来提供更加精准的消费能力。