KafkaAPI 2 API消费者

普通消费者

代码

```
package com.atguigu.kafka.consume;

import java.util.Arrays;
import java.util.Properties;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
public class CustomNewConsumer {
    public static void main(String[] args) {
        Properties props = new Properties();
        // 定义 kakfa 服务的地址，不需要将所有 broker 指定上
        //props.put("bootstrap.servers", "hadoop102:9092");
        props.put(ConsumerConfig.BOOTSTEAP_SERVERS  _CONFIG, "hadoop102:9092");

        // 制定 consumer group
        props.put("group.id", "test");

        // 是否自动确认 offset
        props.put("enable.auto.commit", "true");

        // 自动确认 offset 的时间间隔
        props.put("auto.commit.interval.ms", "1000");

        // key 的序列化类
        props.put("key.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");

        // value 的序列化类
        props.put("value.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");

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

        // 消费者订阅的 topic, 可同时订阅多个
        consumer.subscribe(Arrays.asList("first", "second","third"));
        while (true) {
            // 读取数据，读取超时时间为 100ms
            ConsumerRecords<String, String> records = consumer.poll(100);
            for (ConsumerRecord<String, String> record : records)
                System.out.printf("offset = %d, key = %s, value = %s%n",
record.offset(), record.key(), record.value());

        }
    }
```

```
}
```

- 先运行生产者然后是消费者。

重置offset

> 特定场景下生效：
>
> 1. 没有初始化的offset，当前消费者组第一次消费
> 2. 当前的offset已经不存在，数据被删除

> 两个参数可选:
>
> 1. earlist：
> 2. latest: 最新的数据

```
//重置消费者的offset
pros.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
```

关闭和手动提交offset

```
// 是否自动确认 offset ,内存中仍然可以进行offset，但是结束后不会写回到kafka中
props.put("enable.auto.commit", "false");
```

> 虽然自动提交 offset 十分简洁便利，但是由于是基于时间提交的，开发人员难以把握offset的提交时机。因此Kafka还提供了手动提交的offset的API。
>
> 手动提交的offset的方法有两种，分别是 commitSync (同步提交) 和 commitAsync(异步提交)。两者的相同点是，都会将本次poll的一批数据最高的偏移量提交。不同点是，commitSync阻塞当前线程，一直到提交成功，并且会自动失败重试（由不可控因素导致，也会出现提交失败）；而commitAsync则没有失败重试机制，故有可能提交失败。

同步提交

```
// 关闭自动提交
props.put("enable.auto.commit", "false");


//程序末尾,同步提交，当前线程会阻塞到offset提交成功
consumer.commitSync();
```

异步提交

```
//程序末尾
comsumer.commitAsync(new OffsetCommitCallback(){
    @Override
    public void onComplete(Map<TopicPartition, OffsetAndMetadata>offsets,
Exception exception){
        if(exception != null){
            System.err.println("Commit failed for"+ offsets);
        }
    }
});
```

同步提交和异步提交都会导致数据重复。所以官方提供了一个方法，自定义存储offset。

自定义存储offset

> Kafka0.9 版本之前，offset 存储在zookeeper，0.9版本及之后，默认存储在Kafka的一个内置的topic中，除此之外，Kafka还可以选择自定义存储offset。
>
> offset的维护是相当繁琐的，因为需要考虑到消费者的Rebalace
>
> **当有新的消费者加入消费者组，已有的消费者推出消费者组或者所订阅的主题和分区发送变化，就会触发到分区的重新分配，重新分配的过程叫做Rebalance**
>
> 消费者发生Rebalance之后，每个消费者消费的分区就会发生变化，因此消费者首先获取自己被重新分配到的分区，并且定位到每个分区最近提交的offset位置继续消费。
>
> 要实现自定义存储offset，需要借助ConsumerRebalanceListener，以下为示例代码，其中提交和获取offset的方法，需要根据所选的offset存储系统自行实现。

```
package com.atguigu.kafka.consume;

import java.util.Arrays;
import java.util.Properties;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
public class CustomNewConsumer {
    public static void main(String[] args) {
        Properties props = new Properties();
        // 定义 kakfa 服务的地址，不需要将所有 broker 指定上
        //props.put("bootstrap.servers", "hadoop102:9092");
        props.put(ConsumerConfig.BOOTSTEAP_SERVERS  _CONFIG, "hadoop102:9092");

        // 制定 consumer group
        props.put("group.id", "test");

        // 是否自动确认 offset
        props.put("enable.auto.commit", "true");

        // 自动确认 offset 的时间间隔
        props.put("auto.commit.interval.ms", "1000");

        // key 的序列化类
        props.put("key.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");
```

```
        // value 的序列化类
        props.put("value.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");

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

        //消费者订阅主题
        consumer.subscribe(Arrays.asList("first"), new
ConsumerRebalanceListener(){

        //该方法再Rebalance之前调用
        @Override
        public onPartitionRevoked(Collection<TopicPartition> partition){

        commitOffset(currentOffset);

        }

        //该方法在Rebanlance之后调用
        @Orride
        public onPartitionsAssigned(Collection<TopicPartition> partition){
            currentOffset.clear();
            for(TopicPartition partition : partitions){
                consumer.seek(partition, getOffset(partition));//定位到最近提交的消
费者
            }
        }});
         while (true) {
            // 读取数据，读取超时时间为 100ms
            ConsumerRecords<String, String> records = consumer.poll(100);
            for (ConsumerRecord<String, String> record : records)
                System.out.printf("offset = %d, key = %s, value = %s%n",
record.offset(), record.key(), record.value());
             currentOffset.put(new TopicPartition(record.topic(),
record.partition()), record.offset());
        }
        commitOffset(currentOffset);//异步提交
    }

    //获取某分区的最新offset
    private static long getOffset(TopicPartition partition){
        return 0;
    }

    //提交该消费者所有分区的offset
    private static void commitOffset(Map<TopicPartition, Long> currentOffset){
}
}
```