第 4 章 Kafka API 实战

4.1 环境准备

1）在 eclipse 中创建一个 java 工程

2）在工程的根目录创建一个 lib 文件夹

3)解压 kafka 安装包，将安装包 libs 目录下的 jar 包拷贝到工程的 lib 目录下，并 build path。

4）启动 zk 和 kafka 集群，在 kafka 集群中打开一个消费者

```
[atguigu@hadoop102 kafka]$ bin/kafka-console-consumer.sh --zookeeper hadoop102:2181
--topic first
```

4.2 Kafka 生产者 Java API

4.2.1 创建生产者（过时的 API）

```
package com.atguigu.kafka;
import java.util.Properties;
import kafka.javaapi.producer.Producer;
import kafka.producer.KeyedMessage;
import kafka.producer.ProducerConfig;

public class OldProducer {

    @SuppressWarnings("deprecation")
    public static void main(String[] args) {

        Properties properties = new Properties();
        properties.put("metadata.broker.list", "hadoop102:9092");
        properties.put("request.required.acks", "1");
        properties.put("serializer.class", "kafka.serializer.StringEncoder");

        Producer<Integer, String> producer = new Producer<Integer,String>(new
ProducerConfig(properties));

        KeyedMessage<Integer, String> message = new KeyedMessage<Integer,
String>("first", "hello world");
        producer.send(message );
    }
}
```

4.2.2 创建生产者（新 API）

```
package com.atguigu.kafka;
import java.util.Properties;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerRecord;

public class NewProducer {

	public static void main(String[] args) {

		Properties props = new Properties();
		// Kafka 服务端的主机名和端口号
		props.put("bootstrap.servers", "hadoop103:9092");
		// 等待所有副本节点的应答
		props.put("acks", "all");
		// 消息发送最大尝试次数
		props.put("retries", 0);
		// 一批消息处理大小
		props.put("batch.size", 16384);
		// 请求延时
		props.put("linger.ms", 1);
		// 发送缓存区内存大小
		props.put("buffer.memory", 33554432);
		// key 序列化
		props.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
		// value 序列化
		props.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");

		Producer<String, String> producer = new KafkaProducer<>(props);
		for (int i = 0; i < 50; i++) {
			producer.send(new ProducerRecord<String, String>("first", Integer.toString(i),
"hello world-" + i));
		}

		producer.close();
	}
}
```

4.2.3 创建生产者带回调函数（新 API）

```
package com.atguigu.kafka;
import java.util.Properties;
import org.apache.kafka.clients.producer.Callback;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;

public class CallBackProducer {

    public static void main(String[] args) {

        Properties props = new Properties();
        // Kafka 服务端的主机名和端口号
        props.put("bootstrap.servers", "hadoop103:9092");
        // 等待所有副本节点的应答
        props.put("acks", "all");
        // 消息发送最大尝试次数
        props.put("retries", 0);
        // 一批消息处理大小
        props.put("batch.size", 16384);
        // 增加服务端请求延时
        props.put("linger.ms", 1);
        // 发送缓存区内存大小
        props.put("buffer.memory", 33554432);
        // key 序列化
        props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        // value 序列化
        props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

        KafkaProducer<String, String> kafkaProducer = new KafkaProducer<>(props);

        for (int i = 0; i < 50; i++) {

            kafkaProducer.send(new ProducerRecord<String, String>("first", "hello" + i), new
Callback() {

                @Override
                public void onCompletion(RecordMetadata metadata, Exception exception) {

                    if (metadata != null) {

                        System.err.println(metadata.partition() + "---" + metadata.offset());
```

```
                        }
                    }
                });
            }

            kafkaProducer.close();
        }
}
```

4.2.4 自定义分区生产者

0）需求：将所有数据存储到 topic 的第 0 号分区上

1）定义一个类实现 Partitioner 接口，重写里面的方法（过时 API）

```
package com.atguigu.kafka;
import java.util.Map;
import kafka.producer.Partitioner;

public class CustomPartitioner implements Partitioner {

    public CustomPartitioner() {
        super();
    }

    @Override
    public int partition(Object key, int numPartitions) {
        // 控制分区
        return 0;
    }
}
```

2）自定义分区（新 API）

```
package com.atguigu.kafka;
import java.util.Map;
import org.apache.kafka.clients.producer.Partitioner;
import org.apache.kafka.common.Cluster;

public class CustomPartitioner implements Partitioner {

    @Override
    public void configure(Map<String, ?> configs) {

    }

    @Override
```

```
    public int partition(String topic, Object key, byte[] keyBytes, Object value, byte[] valueBytes, Cluster cluster) {
        // 控制分区
        return 0;
    }

    @Override
    public void close() {

    }
}
```

3）在代码中调用

```
package com.atguigu.kafka;
import java.util.Properties;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerRecord;

public class PartitionerProducer {

    public static void main(String[] args) {

        Properties props = new Properties();
        // Kafka 服务端的主机名和端口号
        props.put("bootstrap.servers", "hadoop103:9092");
        // 等待所有副本节点的应答
        props.put("acks", "all");
        // 消息发送最大尝试次数
        props.put("retries", 0);
        // 一批消息处理大小
        props.put("batch.size", 16384);
        // 增加服务端请求延时
        props.put("linger.ms", 1);
        // 发送缓存区内存大小
        props.put("buffer.memory", 33554432);
        // key 序列化
        props.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
        // value 序列化
        props.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
        // 自定义分区
        props.put("partitioner.class", "com.atguigu.kafka.CustomPartitioner");
```

```
        Producer<String, String> producer = new KafkaProducer<>(props);
        producer.send(new ProducerRecord<String, String>("first", "1", "atguigu"));

        producer.close();
    }
}
```

4）测试

（1）在 hadoop102 上监控/opt/module/kafka/logs/目录下 first 主题 3 个分区的 log 日志动态变化情况

```
[atguigu@hadoop102 first-0]$ tail -f 00000000000000000000.log
[atguigu@hadoop102 first-1]$ tail -f 00000000000000000000.log
[atguigu@hadoop102 first-2]$ tail -f 00000000000000000000.log
```

（2）发现数据都存储到指定的分区了。

4.3 Kafka 消费者 Java API

0）在控制台创建发送者

```
[atguigu@hadoop104 kafka]$ bin/kafka-console-producer.sh --broker-list hadoop102:9092
--topic first
>hello world
```

1）创建消费者（过时 API）

```
package com.atguigu.kafka.consume;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import kafka.consumer.Consumer;
import kafka.consumer.ConsumerConfig;
import kafka.consumer.ConsumerIterator;
import kafka.consumer.KafkaStream;
import kafka.javaapi.consumer.ConsumerConnector;

public class CustomConsumer {

    @SuppressWarnings("deprecation")
    public static void main(String[] args) {
        Properties properties = new Properties();
```

```
            properties.put("zookeeper.connect", "hadoop102:2181");
            properties.put("group.id", "g1");
            properties.put("zookeeper.session.timeout.ms", "500");
            properties.put("zookeeper.sync.time.ms", "250");
            properties.put("auto.commit.interval.ms", "1000");

            // 创建消费者连接器
            ConsumerConnector consumer = Consumer.createJavaConsumerConnector(new ConsumerConfig(properties));

            HashMap<String, Integer> topicCount = new HashMap<>();
            topicCount.put("first", 1);

            Map<String, List<KafkaStream<byte[], byte[]>>> consumerMap = consumer.createMessageStreams(topicCount);

            KafkaStream<byte[], byte[]> stream = consumerMap.get("first").get(0);

            ConsumerIterator<byte[], byte[]> it = stream.iterator();

            while (it.hasNext()) {
                System.out.println(new String(it.next().message()));
            }
        }
}
```

2）官方提供案例（自动维护消费情况）（新 API）

```
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");
        // 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());
            }
        }
}`
```