kafka 深入

kafka hadoop

配置sl4j日志

1. 导入sl4j依赖

```
<dependency>
 <groupId>org.slf4j</groupId>
 <artifactId>slf4j-log4j12</artifactId>
 <version>1.7.21</version>
</dependency>
```

2. 添加log4j文件

java API 创建Producer

1. 创建连接

```
/**
* 创建一个新的实例 ProducerClient.
* 构造方法
*/
public ProducerClient() {
 properties = new Properties();
 properties.put("bootstrap.servers", "master:9092,slaver1:9092");
 properties.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
 properties.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
 producer = new KafkaProducer<>(properties);
}
```

2. 发送数据

```
/**
* sendRecorder 发送数据
* @param @param key
* @param @param value 参数
* @return void 返回类型
* @Exception 异常对象
* @author Allen
*/
```

```
public void sendRecorder(String key, String value) {
 ProducerRecord<String, String> record = new ProducerRecord<>("from-java", key, value)
;
 producer.send(record);
}
```

3. 关闭连接

```
/**
* close 刷新数据, 关闭连接
* @param  参数
* @return void 返回类型
* @Exception 异常对象
* @author Allen
*/
public void close() {
 producer.flush();
 producer.close();
}
```

4. 指定分区发送数据

```
/**
* assignPartitionSend 指定分区发送数据
* @param @param key
* @param @param value 参数
* @return void 返回类型
* @Exception 异常对象
* @author Allen
*/
public void assignPartitionSend(String key, String value) {
 ProducerRecord<String, String> record = new ProducerRecord<>("from-java", 0, key, val
ue);
 producer.send(record);
}
```

5. 获取topic详细信息

```
/**
* getTopicPartition 获取topic的详细信息
* @param @param topic 参数
* @return void 返回类型
* @Exception 异常对象
* @author Allen
*/
public void getTopicPartition(String topic) {

 List<PartitionInfo> partitionInfos = producer.partitionsFor(topic);
 for (PartitionInfo partitionInfo : partitionInfos) {
  System.out.println(partitionInfo);
 }
}
```

6. 获取集群状态信息

```
/**
* getMetrics 获取集群状态信息
* @param  参数
```

```
* @return void 返回类型
* @Exception 异常对象
* @author Allen
*/
public void getMetrics() {
 Map<MetricName, ? extends Metric> metrics = producer.metrics();
 for (MetricName name : metrics.keySet()) {
  System.out.println(name.name() + " : " + metrics.get(name).value());
 }
}
```

7. 发送数据返回状态信息---回调函数

```
 /**
 * sendRecorderWithCallback 回调函数返回发送信息
 * @param @param key
 * @param @param value 参数
 * @return void 返回类型
 * @Exception 异常对象
 * @author Allen
 */
 public void sendRecorderWithCallback(String key, String value) {

  Logger logger = LoggerFactory.getLogger(ProducerClient.class);
  ProducerRecord<String, String> record = new ProducerRecord<>("from-java", key, value
);
  producer.send(record, new Callback() {
   // 回调方法，在整个方法被调用之后，Record的接收和存储是由服务端来完成
   // callback是服务端接收服务器存储完了，返回回调的一个函数
   @Override
   public void onCompletion(RecordMetadata metadata, Exception exception) {
    if (exception == null) {
     logger.info("存储位置:partition:" + metadata.partition() + ",offset:" + metadata.of
fset()
       + ",timestrap:" + metadata.timestamp());
    } else {
     logger.warn("服务端出现异常: ");
     exception.printStackTrace();
    }
   }
  });
 }
```

java API 创建Consumer

1. 创建连接

```
/**
* 创建一个新的实例 ConsumerClient.
* 构造方法
*/
public ConsumerClient() {
 properties = new Properties();
 properties.put("bootstrap.servers", "master:9092,slaver1:9092");
 properties.put("group.id", "java_group");
 properties.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeser
ializer");
 properties.put("value.deserializer", "org.apache.kafka.common.serialization.StringDes
erializer");
 consumer = new KafkaConsumer<>(properties);
}
```

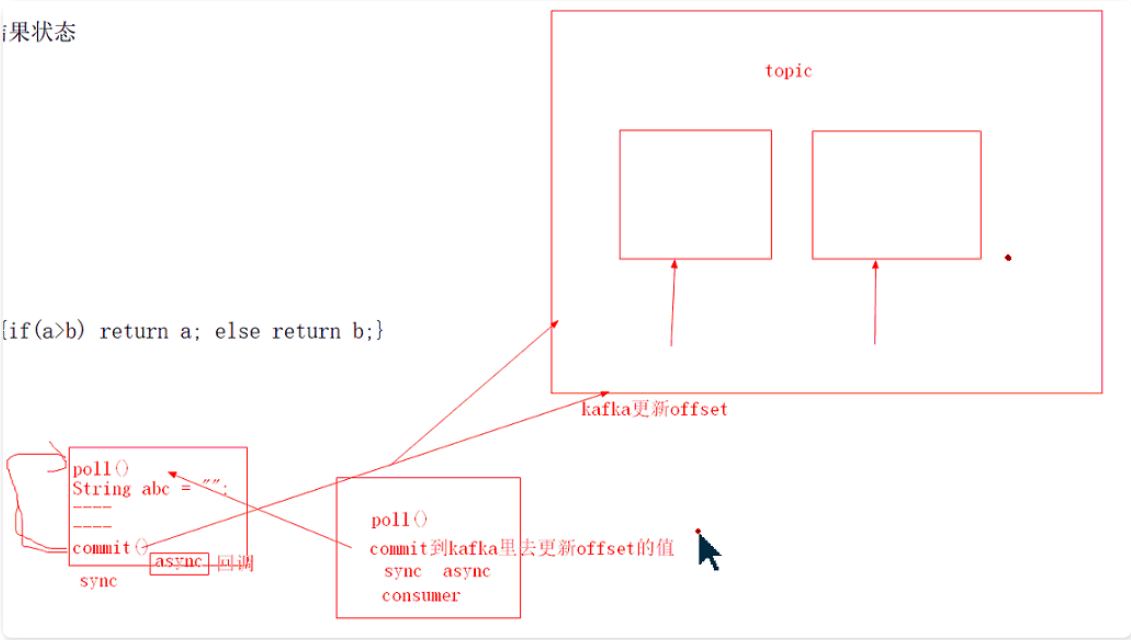
2. 订阅topic方法

```
/**
* subscribeTopic 订阅topic
* @param  参数
* @return void 返回类型
* @Exception 异常对象
* @author Allen
*/
public void subscribeTopic(){
 consumer.subscribe(Arrays.asList("from-java"));
 while(true){
  // 从kafka拉取数据
  ConsumerRecords<String,String> records = consumer.poll(1000);

  for (ConsumerRecord<String, String> record : records) {
   System.out.printf("partition = %d, offset = %d, key = %s, value = %s%n",
     record.partition(), record.offset(), record.key(), record.value());
  }
 }
}
```

同步异步

同步：提交请求--->等待服务器处理--->处理完毕返回 这个期间不能poll取其他的数据
异步:请求通过事件触发--->服务器处理（这段时间还继续poll数据）--->处理完毕



消费者手动提交

1. 创建连接

```
public ManualCommitConsumer() {
  properties.setProperty("bootstrap.servers", "master:9092,slaver1:9092");
  properties.setProperty("key.deserializer", "org.apache.kafka.common.serialization.St
ringDeserializer");
  properties.setProperty("value.deserializer", "org.apache.kafka.common.serialization.
StringDeserializer");
```

```
  properties.setProperty("group.id", "java_group");
  // 取消自动提交
  properties.setProperty("enable.auto.commit", "false");
  properties.setProperty("auto.offset.reset", "none");
  consumer = new KafkaConsumer<>(properties);
 }
```

2. 订阅topic

```
public void substituteTopic() {
 List<String> topics = new ArrayList<>();
 topics.add("from-java");
 consumer.subscribe(topics);
 while (true) {
  ConsumerRecords<String, String> records = consumer.poll(1000);
  for (ConsumerRecord<String, String> record : records) {
   System.out.println("partiton = " + record.partition() + " ,offset = " + record.offs
et() + ",key = "
     + record.key() + ",value = " + record.value());
  }
  consumer.commitSync();
 }
}
```

3. 获取topic指定分区上的offset

```
/**
* getOffsets 获取topic指定分区上的offset
* @param  参数
* @return void 返回类型
* @Exception 异常对象
* @author Allen
*/
public void getOffsets() {
 OffsetAndMetadata offsets = consumer.committed(new TopicPartition("from-java", 1));
 System.out.println("offsets = " + offsets.offset());
}
```

4. 指定分区消费，指定从offset的值出开始消费

```
/**
* consumerAssigned 指定分区消费，指定从offset的值出开始消费
* 消费者对topic的消费有两种方式
* 1. consumer.subscribe(topics)
* 2.consumer.assign(topicPartitions);
* 两种方式互斥，任选其一
* @param  参数
* @return void 返回类型
* @Exception 异常对象
* @author Allen
*/
public void consumerAssigned() {
 /*
  * List<String> topics = new ArrayList<>(); topics.add("from-java");
  * consumer.subscribe(topics);
  */
 // 指定分区
 List<TopicPartition> topicPartitions = new ArrayList<TopicPartition>();
 topicPartitions.add(new TopicPartition("from-java", 1));
 consumer.assign(topicPartitions);
 // 指定分区的offset位置
```

```
 TopicPartition partition = new TopicPartition("from-java", 1);
 consumer.seekToEnd(Arrays.asList(partition));
 while (true) {
  ConsumerRecords<String, String> records = consumer.poll(1000);
  for (ConsumerRecord<String, String> record : records) {
   System.out.println("partition = " + record.partition() + ",offset = " + record.offs
et() + ",key = "
     + record.key() + ",value = " + record.value());
  }
 }
}
```

5. 设置offset

```
/**
* setCommitoffset 设置offset
* @param  参数
* @return void 返回类型
* @Exception 异常对象
* @author Allen
*/
public void setCommitoffset() {
 Map<TopicPartition, OffsetAndMetadata> offsets = new HashMap<>();
 offsets.put(new TopicPartition("from-java", 1), new OffsetAndMetadata(830));
 List<String> topics = new ArrayList<>();
 topics.add("from-java");
 consumer.subscribe(topics);
 // 指定位置提交某个分区的offset的值, 这个会在下一次拉取数据之前生效

 while (true) {
  ConsumerRecords<String, String> records = consumer.poll(1000);
  for (ConsumerRecord<String, String> record : records) {
   if(record.partition() == 1){
    System.out.println("offset = " + record.offset() + " ,partition = " + record.parti
tion() + "  ,key = "
      + record.key() + " ,value = " + record.value());
   }
   consumer.commitSync(offsets);
  }

 }
}
```

flume 发送数据，kafka消费

需求：写一个flume客户端，发送数据给avro的flume，然后sink给kafka
写一个kafka的consumer从kafka中消费 flume客户端发送过来的数据

数据流向
flume客户端java程序发送数据---->flume（flume.conf程序）---->kafka----->kafkaconsumer的java程序消费

- flume客户端程序

```
package top.xiesen.flume;

import java.nio.charset.Charset;

import org.apache.flume.Event;
import org.apache.flume.EventDeliveryException;
import org.apache.flume.api.RpcClient;
import org.apache.flume.api.RpcClientFactory;
```

```
import org.apache.flume.event.EventBuilder;


/**
* 项目名称：kafkaclient
* 类名称：FlumeAvroClient
* 类描述：发送avro文件到flume source
* @author Allen
*/
public class FlumeAvroClient {

 private RpcClient flumeClient;
 private String hostname;
 private int port;

 public FlumeAvroClient() {
  super();
 }


 /**
 * 创建一个新的实例 FlumeAvroClient.
 * 实例化 flumeClient
 * @param hostname
 * @param port
 */
 public FlumeAvroClient(String hostname, int port) {
  this.hostname = hostname;
  this.port = port;
  flumeClient = RpcClientFactory.getDefaultInstance(hostname, port);
 }


 /**
 * sendEvent 发送数据
 * @param @param msg 参数
 * @return void 返回类型
 * @Exception 异常对象
 * @author Allen
 */
 public void sendEvent(String msg) {
  Event event = EventBuilder.withBody(msg, Charset.forName("UTF-8"));
  try {
   flumeClient.append(event);
  } catch (EventDeliveryException e) {
   e.printStackTrace();
   flumeClient.close();
   flumeClient = null;
   flumeClient = RpcClientFactory.getDefaultInstance(hostname, port);
  }
 }


 /**
 * close 关闭连接
 * @param  参数
 * @return void 返回类型
 * @Exception 异常对象
 * @author Allen
 */
 public void close(){
  flumeClient.close();
 }


 /**
 * main 测试用例
 * @param @param args 参数
 * @return void 返回类型
```

```
 * @Exception 异常对象
 * @author Allen
 */
 public static void main(String[] args) {
  FlumeAvroClient fac = new FlumeAvroClient("master",9999);
  String msg = "flume_avro_kafka_";

  for(int i = 0; i < 100; i++){
   fac.sendEvent(msg + i);
  }

  fac.close();
 }
}
```

- flume.conf程序

```
a1.sources = r1
a1.sinks=s1
a1.channels=c1

a1.sources.r1.type = avro
a1.sources.r1.bind = master
a1.sources.r1.port = 9999

a1.channels.c1.type= memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100

a1.sinks.s1.type = org.apache.flume.sink.kafka.KafkaSink
a1.sinks.s1.kafka.bootstrap.servers=master:9092
a1.sinks.s1.kafka.topic=flume_kafka
a1.sinks.s1.kafka.flumeBatchSize = 20

a1.sources.r1.channels = c1
a1.sinks.s1.channel = c1
```

- kafkaConsumer消费程序

```
package top.xiesen.bd14;

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 FlumeConsumer {

 private KafkaConsumer<String, String> consumer;
 private Properties properties;

 /**
 * 创建一个新的实例 ConsumerClient.
 * 构造方法
 */
 public FlumeConsumer() {
  properties = new Properties();
  properties.put("bootstrap.servers", "master:9092,slaver1:9092");
  properties.put("group.id", "java_group");
  properties.put("key.deserializer", "org.apache.kafka.common.serialization.StringDese
rializer");
  properties.put("value.deserializer", "org.apache.kafka.common.serialization.StringDe
```

```
serializer");
  consumer = new KafkaConsumer<>(properties);
 }


 /**
 * subscribeTopicFromFlume 订阅flume topic
 * @param  参数
 * @return void 返回类型
 * @Exception 异常对象
 * @author Allen
 */
 public void subscribeTopicFromFlume(){
  consumer.subscribe(Arrays.asList("flume_kafka"));

  while(true){
   ConsumerRecords<String,String> records = consumer.poll(1000);
   for (ConsumerRecord<String, String> record : records) {
    System.out.println("key = " + record.key() + " ,value = " + record.value());
   }
  }
 }


 /**
 * main 测试用例
 * @param @param args 参数
 * @return void 返回类型
 * @Exception 异常对象
 * @author Allen
 */
 public static void main(String[] args) {
  FlumeConsumer fc = new FlumeConsumer();
  fc.subscribeTopicFromFlume();
 }
}
```