Kafka

# 准备

采用最简单的部署方式。

## 下载安装包

官网下载：http://kafka.apache.org/downloads.html

## 解压

# tar -zxvf kafka\_2.10-0.10.0.0.tgz

## 启动内置zookeeper

# bin/zookeeper-server-start.sh config/zookeeper.properties

## 启动kafka server

# bin/kafka-server-start.sh config/server.properties

## 创建topic

# bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test

|  |
| --- |
| #确认创建成功  bin/kafka-topics.sh --list --zookeeper localhost:2181 |

## 开启producer往kakfa发送消息

# bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test

|  |
| --- |
|  |

## 开启consumer从kafka消费消息

# bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic test --from-beginning

|  |
| --- |
|  |

# Demo

## 生产者Producer

KafkaProducer：用于产生消息并往kafka中发送。

|  |
| --- |
| package com.frontsurf;  import com.frontsurf.util.Conf; import kafka.javaapi.producer.Producer; import kafka.producer.KeyedMessage; import kafka.producer.ProducerConfig;  import java.util.Properties;  */\*\*  \* Kafka消息生产者  \* Created by Liupeng on 2016/8/9.  \*/* public class KafkaProducer {  private final Producer<Integer, String> producer;  */\*\*  \* 全局配置  \*/* private Conf conf = Conf.*getInstance*();   */\*\*  \* 构造方法中初始化 producer  \*/* public KafkaProducer(){  Properties props = new Properties();  //配置kafka broker  props.put("metadata.broker.list", conf.getKafkaBrokeList());  //序列化类  props.put("serializer.class", "kafka.serializer.StringEncoder");  producer = new Producer<Integer, String>(new ProducerConfig(props));  }   */\*\*  \* 根据指定消息数目，产生消息往kafka中发送  \** ***@param*** *msgNum  \*/* public void produce(int msgNum){  int count = 1;  while (count<=msgNum){  String msg = "This is a test message,number is \t"+ (count++);  producer.send(new KeyedMessage<Integer, String>(conf.getTopic(), msg));  try{  //延迟发送，模拟真实场景  Thread.*sleep*(conf.getSleep\_time());  }catch (Throwable e){  //do nothing  }  System.*out*.println("Send message------"+msg);  }  }   public static void main(String[] args) {  System.*out*.println("------Producer start produce message------");  KafkaProducer kafkaProducer = new KafkaProducer();  kafkaProducer.produce(kafkaProducer.conf.getMessageNum());  } } |

## 消费者Consumer——Java API

从kafka中消费消息。

|  |
| --- |
| package com.frontsurf;  import com.frontsurf.util.Conf; import kafka.consumer.ConsumerConfig; import kafka.consumer.ConsumerIterator; import kafka.consumer.KafkaStream; import kafka.javaapi.consumer.ConsumerConnector; import kafka.message.MessageAndMetadata;  import java.util.HashMap; import java.util.List; import java.util.Map; import java.util.Properties;  */\*\*  \* Kafka消息消费者  \* Created by Liupeng on 2016/8/9.  \*/* public class KafkaConsumer {  private final ConsumerConnector consumer;  */\*\*  \* 全局配置  \*/* private Conf conf = Conf.*getInstance*();   */\*\*  \* 构造方法中初始化 consumer  \*/* public KafkaConsumer(){  Properties props = new Properties();  //配置zookeeper  props.put("zookeeper.connect", conf.getZkHosts());  //配置消费组  props.put("group.id", conf.getGroupId());  //序列化类  props.put("serializer.class", "kafka.serializer.StringEncoder");  props.put("zookeeper.session.timeout.ms", "4000");  props.put("zookeeper.sync.time.ms", "200");  props.put("auto.commit.interval.ms", "1000");   ConsumerConfig config = new ConsumerConfig(props);  consumer = kafka.consumer.Consumer.*createJavaConsumerConnector*(config);  }   */\*\*  \* 从kafka中消费消息  \*/* public void consume(){  Map<String, Integer> topicCountMap = new HashMap<String, Integer>();  topicCountMap.put(conf.getTopic(), new Integer(1));   Map<String, List<KafkaStream<byte[], byte[]>>> messageStreams = consumer.createMessageStreams(topicCountMap);  KafkaStream<byte[], byte[]> stream = messageStreams.get(conf.getTopic()).get(0);  ConsumerIterator<byte[], byte[]> iterator = stream.iterator();  while (iterator.hasNext()){  MessageAndMetadata<byte[], byte[]> data = iterator.next();  System.*out*.println("Get message------"+new String(data.message()));  }  }   public static void main(String[] args) {  System.*out*.println("------Consumer start consume message------");  new KafkaConsumer().consume();  } } |

## 消费者Consumer——Spark API

通过KafkaUtils创建streaming，流式读取kafka消息。

|  |
| --- |
| **package** com.frontsurf  **import** com.frontsurf.util.Conf **import** kafka.serializer.StringDecoder **import** org.apache.spark.SparkConf **import** org.apache.spark.streaming.kafka.KafkaUtils **import** org.apache.spark.streaming.{Durations, StreamingContext}  */\*\*  \* Spark流式消费Kafka中消息  \* Created by Liupeng on 2016/8/9.  \*/* **object** SparkKafkaConsumer {  */\*\*  \* 全局配置  \*/* **private val** *conf* = Conf.*getInstance*()   **def** main(args: Array[String]) {  **val** config = **new** SparkConf().setAppName("SparkKafkaConsumer").setMaster("local")  //定义流处理，每5秒一次微批处理  **val** ssc = **new** StreamingContext(config,Durations.*seconds*(*conf*.getInterval))  //配置kafka broker和消费组  **val** map = *Map*(("metadata.broker.list", *conf*.getKafkaBrokeList),("group.id", *conf*.getGroupId))  //配置topic  **val** set = *Set*(*conf*.getTopic)  //使用spark提供的KafkaUtils，在Kafka上建立流，持续消费kafka中消息  **val** lines = KafkaUtils.*createDirectStream*[String,String,StringDecoder,StringDecoder](ssc,map,set)  *println*("------Spark Consumer start consume message------")  //处理并打印消息  lines.map("Get message------"+\_.\_2).foreachRDD(\_.foreach(*println*))  ssc.start()  ssc.awaitTermination()  } } |

## 配置加载工具

通过加载属性配置文件，实现动态配置的功能。

|  |
| --- |
| package com.frontsurf.util;  import java.io.FileInputStream; import java.io.FileNotFoundException; import java.io.IOException; import java.util.Properties; */\*\*  \* 单例类,保存全局配置  \** ***@author*** *Liupeng  \*  \*/* public class Conf {  /\*  \* 单例模式  \*/  private static final Conf *conf* = new Conf();   private final String zkHosts;  private final String groupId;  private final String topic;  private final String kafkaBrokeList;  private int interval;  private int messageNum;  private int sleep\_time;   private Conf() {  /\* IDE上测试 \*/ // String baseDirectory = System.getProperty("user.dir"); // String confFile = baseDirectory + "/config/conf.properties";  /\* 打包测试，需在conf.properties文件所在目录运行程序 \*/  String confFile = "./conf.properties";  Properties prop = new Properties();  try {  FileInputStream in = new FileInputStream(confFile);  prop.load(in);  } catch (FileNotFoundException e) {  throw new ExceptionInInitializerError(e);  } catch (IOException e) {  throw new ExceptionInInitializerError(e);  }  zkHosts = prop.getProperty("zkHosts");  groupId = prop.getProperty("groupId");  topic = prop.getProperty("topic");  kafkaBrokeList = prop.getProperty("kafkaBrokeList");  interval = Integer.*parseInt*(prop.getProperty("interval","5"));  messageNum = Integer.*parseInt*(prop.getProperty("messageNum","1000"));  sleep\_time = Integer.*parseInt*(prop.getProperty("sleep\_time","100"));  }   */\*\*  \* 返回单例类实例  \** ***@return*** *\*/* public static Conf getInstance() {  return *conf*;  }   public String getZkHosts() {  return zkHosts;  }   public String getGroupId() {  return groupId;  }   public String getTopic() {  return topic;  }   public String getKafkaBrokeList() {  return kafkaBrokeList;  }   public int getInterval() {  return interval;  }   public int getMessageNum() {  return messageNum;  }   public int getSleep\_time() {  return sleep\_time;  } } |

## 属性配置文件

配置信息存入属性文件，实现动态配置的功能。若是打包测试，属性配置文件需保存在/etc/kafka/conf.properties

|  |
| --- |
| # ===========kafka相关配置=========== topic = test groupId = test-group #kafkaBrokeList = 192.168.100.174:9092,192.168.100.175:9092,192.168.100.176:9092 kafkaBrokeList = 192.168.139.100:9092  # ===========zookeeper相关配置=========== #zkHosts = 192.168.100.171:2181,192.168.100.172:2181,192.168.100.173:2181 zkHosts = 192.168.139.100:2181  # ===========spark相关配置=========== interval = 5  # ===========运行程序相关配置=========== messageNum = 500 sleep\_time = 100 |

## 相关测试命令

Kafka后台启动命令：

# bin/kafka-server-start.sh config/server.properties >/dev/null 2>&1 &

查看topic描述信息：

# bin/kafka-topics.sh --describe --zookeeper node1:2181 --topic test

启动kafka消息生产者（为了演示比较直观，最好单独开启一个窗口）：

# java -cp Kafka\_Demo.jar com.frontsurf.KafkaProducer

启动kafka消息消费者（为了演示比较直观，最好单独开启一个窗口）：

# java -cp Kafka\_Demo.jar com.frontsurf.KafkaConsumer

启动spark kafka消息消费者（为了演示比较直观，最好单独开启一个窗口）：

# java -cp Kafka\_Demo.jar com.frontsurf.SparkKafkaConsumer