参考：<http://blog.csdn.net/lizhitao/article/details/37811291>

**apache kafka系列之客户端开发-java**

apache kafka中国社区QQ群:162272557

1.依赖包

        <dependency>  
            <groupId>org.apache.kafka</groupId>  
            <artifactId>kafka\_2.10</artifactId>  
            <version>0.8.1</version>  
        </dependency>

2.producer程序开发例子

2.1 producer参数说明

#指定kafka节点列表，用于获取metadata，不必全部指定  
metadata.broker.list=192.168.2.105:9092,192.168.2.106:9092  
# 指定分区处理类。默认kafka.producer.DefaultPartitioner，表通过key哈希到对应分区  
#partitioner.class=com.meituan.mafka.client.producer.CustomizePartitioner  
   
# 是否压缩，默认0表示不压缩，1表示用gzip压缩，2表示用snappy压缩。压缩后消息中会有头来指明消息压缩类型，故在消费者端消息解压是透明的无需指定。  
compression.codec=none  
    
# 指定序列化处理类(mafka client API调用说明-->3.序列化约定wiki)，默认为kafka.serializer.DefaultEncoder,即byte[]  
serializer.class=com.meituan.mafka.client.codec.MafkaMessageEncoder  
# serializer.class=kafka.serializer.DefaultEncoder  
# serializer.class=kafka.serializer.StringEncoder  
# 如果要压缩消息，这里指定哪些topic要压缩消息，默认empty，表示不压缩。  
#compressed.topics=  
   
########### request ack ###############  
# producer接收消息ack的时机.默认为0.   
# 0: producer不会等待broker发送ack   
# 1: 当leader接收到消息之后发送ack   
# 2: 当所有的follower都同步消息成功后发送ack.   
request.required.acks=0   
# 在向producer发送ack之前,broker允许等待的最大时间   
# 如果超时,broker将会向producer发送一个error ACK.意味着上一次消息因为某种   
# 原因未能成功(比如follower未能同步成功)   
request.timeout.ms=10000  
########## end #####################  
   
   
# 同步还是异步发送消息，默认“sync”表同步，"async"表异步。异步可以提高发送吞吐量,  
# 也意味着消息将会在本地buffer中,并适时批量发送，但是也可能导致丢失未发送过去的消息  
producer.type=sync  
############## 异步发送 (以下四个异步参数可选) ####################  
# 在async模式下,当message被缓存的时间超过此值后,将会批量发送给broker,默认为5000ms  
# 此值和batch.num.messages协同工作.  
queue.buffering.max.ms = 5000  
# 在async模式下,producer端允许buffer的最大消息量  
# 无论如何,producer都无法尽快的将消息发送给broker,从而导致消息在producer端大量沉积  
# 此时,如果消息的条数达到阀值,将会导致producer端阻塞或者消息被抛弃，默认为10000  
queue.buffering.max.messages=20000  
# 如果是异步，指定每次批量发送数据量，默认为200  
batch.num.messages=500  
# 当消息在producer端沉积的条数达到"queue.buffering.max.meesages"后   
# 阻塞一定时间后,队列仍然没有enqueue(producer仍然没有发送出任何消息)   
# 此时producer可以继续阻塞或者将消息抛弃,此timeout值用于控制"阻塞"的时间   
# -1: 无阻塞超时限制,消息不会被抛弃   
# 0:立即清空队列,消息被抛弃   
queue.enqueue.timeout.ms=-1  
################ end ###############  
   
# 当producer接收到error ACK,或者没有接收到ACK时,允许消息重发的次数   
# 因为broker并没有完整的机制来避免消息重复,所以当网络异常时(比如ACK丢失)   
# 有可能导致broker接收到重复的消息,默认值为3.  
message.send.max.retries=3  
   
   
# producer刷新topic metada的时间间隔,producer需要知道partition leader的位置,以及当前topic的情况   
# 因此producer需要一个机制来获取最新的metadata,当producer遇到特定错误时,将会立即刷新   
# (比如topic失效,partition丢失,leader失效等),此外也可以通过此参数来配置额外的刷新机制，默认值600000   
topic.metadata.refresh.interval.ms=60000

**[plain]** [view plain](http://blog.csdn.net/lizhitao/article/details/37811291) [copy](http://blog.csdn.net/lizhitao/article/details/37811291)

1. import java.util.\*;
3. import kafka.javaapi.producer.Producer;
4. import kafka.producer.KeyedMessage;
5. import kafka.producer.ProducerConfig;
7. public class TestProducer {
8. public static void main(String[] args) {
9. long events = Long.parseLong(args[0]);
10. Random rnd = new Random();
12. Properties props = new Properties();//属性配置
13. props.put("metadata.broker.list", "192.168.2.105:9092");
14. props.put("serializer.class", "kafka.serializer.StringEncoder"); //默认字符串编码消息
15. props.put("partitioner.class", "example.producer.SimplePartitioner");
16. props.put("request.required.acks", "1");
18. ProducerConfig config = new ProducerConfig(props);
20. Producer<String, String> producer = new Producer<String, String>(config);  //producer客户端获取
22. for (long nEvents = 0; nEvents < events; nEvents++) {
23. long runtime = new Date().getTime();
24. String ip = “192.168.2.” + rnd.nextInt(255);
25. String msg = runtime + “,www.example.com,” + ip;
26. KeyedMessage<String, String> data = new KeyedMessage<String, String>("page\_visits", ip, msg);  //topic，
27. producer.send(data);  //发送数据
28. }
29. producer.close();  //关闭
30. }
31. }

2.1 指定关键字key，发送消息到指定partitions

说明：如果需要实现自定义partitions消息发送，需要实现Partitioner接口

**[java]** [view plain](http://blog.csdn.net/lizhitao/article/details/37811291) [copy](http://blog.csdn.net/lizhitao/article/details/37811291)

1. **public** **class** CustomizePartitioner **implements** Partitioner {
2. **public** CustomizePartitioner(VerifiableProperties props) {
4. }
5. /\*\*
6. \* 返回分区索引编号
7. \* @param key sendMessage时，输出的partKey
8. \* @param numPartitions topic中的分区总数
9. \* @return
10. \*/
11. @Override
12. **public** **int** partition(Object key, **int** numPartitions) {
13. System.out.println("key:" + key + "  numPartitions:" + numPartitions);
14. String partKey = (String)key;
15. **if** ("part2".equals(partKey))
16. **return** 2;
17. //        System.out.println("partKey:" + key);
19. ........
20. ........
21. **return** 0;
22. }
23. }

3.consumer程序开发例子

3.1 consumer参数说明

# zookeeper连接服务器地址，此处为线下测试环境配置(kafka消息服务-->kafka broker集群线上部署环境wiki)  
# 配置例子："127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002"  
zookeeper.connect=192.168.2.225:2181,192.168.2.225:2182,192.168.2.225:2183/config/mobile/mq/mafka  
# zookeeper的session过期时间，默认5000ms，用于检测消费者是否挂掉，当消费者挂掉，其他消费者要等该指定时间才能检查到并且触发重新负载均衡  
zookeeper.session.timeout.ms=5000  
zookeeper.connection.timeout.ms=10000  
#当consumer reblance时，重试失败时时间间隔。

zookeeper.sync.time.ms=2000  
   
#指定消费组  
group.id=xxx  
# 当consumer消费一定量的消息之后,将会自动向zookeeper提交offset信息   
# 注意offset信息并不是每消费一次消息就向zk提交一次,而是现在本地保存(内存),并定期提交,默认为true  
auto.commit.enable=true  
# 自动更新时间。默认60 \* 1000  
auto.commit.interval.ms=1000  
   
# 当前consumer的标识,可以设定,也可以有系统生成,主要用来跟踪消息消费情况,便于观察  
conusmer.id=xxx   
   
# 消费者客户端编号，用于区分不同客户端，默认客户端程序自动产生  
client.id=xxxx  
# 最大取多少块缓存到消费者(默认10)  
queued.max.message.chunks=50  
# 当有新的consumer加入到group时,将会reblance,此后将会有partitions的消费端迁移到新   
# 的consumer上,如果一个consumer获得了某个partition的消费权限,那么它将会向zk注册   
# "Partition Owner registry"节点信息,但是有可能此时旧的consumer尚没有释放此节点,   
# 此值用于控制,注册节点的重试次数.   
rebalance.max.retries=5  
# 获取消息的最大尺寸,broker不会像consumer输出大于此值的消息chunk  
# 每次feth将得到多条消息,此值为总大小,提升此值,将会消耗更多的consumer端内存  
fetch.min.bytes=6553600  
# 当消息的尺寸不足时,server阻塞的时间,如果超时,消息将立即发送给consumer  
fetch.wait.max.ms=5000  
socket.receive.buffer.bytes=655360  
   
# 如果zookeeper没有offset值或offset值超出范围。那么就给个初始的offset。有smallest、largest、  
# anything可选，分别表示给当前最小的offset、当前最大的offset、抛异常。默认largest  
auto.offset.reset=smallest  
# 指定序列化处理类(mafka client API调用说明-->3.序列化约定wiki)，默认为kafka.serializer.DefaultDecoder,即byte[]  
derializer.class=com.meituan.mafka.client.codec.MafkaMessageDecoder

3.2 多线程并行消费topic

ConsumerTest类

**[java]** [view plain](http://blog.csdn.net/lizhitao/article/details/37811291) [copy](http://blog.csdn.net/lizhitao/article/details/37811291)

1. **import** kafka.consumer.ConsumerIterator;
2. **import** kafka.consumer.KafkaStream;
4. **public** **class** ConsumerTest **implements** Runnable {
5. **private** KafkaStream m\_stream;
6. **private** **int** m\_threadNumber;
8. **public** ConsumerTest(KafkaStream a\_stream, **int** a\_threadNumber) {
9. m\_threadNumber = a\_threadNumber;
10. m\_stream = a\_stream;
11. }
13. **public** **void** run() {
14. ConsumerIterator<**byte**[], **byte**[]> it = m\_stream.iterator();  //多个消费者
15. **while** (it.hasNext())
16. System.out.println("Thread " + m\_threadNumber + ": " + **new** String(it.next().message()));
17. System.out.println("Shutting down Thread: " + m\_threadNumber);
18. }
19. }

ConsumerGroupExample类

**[java]** [view plain](http://blog.csdn.net/lizhitao/article/details/37811291) [copy](http://blog.csdn.net/lizhitao/article/details/37811291)

1. **import** kafka.consumer.ConsumerConfig;
2. **import** kafka.consumer.KafkaStream;
3. **import** kafka.javaapi.consumer.ConsumerConnector;
5. **import** java.util.HashMap;
6. **import** java.util.List;
7. **import** java.util.Map;
8. **import** java.util.Properties;
9. **import** java.util.concurrent.ExecutorService;
10. **import** java.util.concurrent.Executors;
12. **public** **class** ConsumerGroupExample {
13. **private** **final** ConsumerConnector consumer;
14. **private** **final** String topic;
15. **private**  ExecutorService executor;
17. **public** ConsumerGroupExample(String a\_zookeeper, String a\_groupId, String a\_topic) {
18. consumer = kafka.consumer.Consumer.createJavaConsumerConnector(
19. createConsumerConfig(a\_zookeeper, a\_groupId));//创建消费组
20. **this**.topic = a\_topic;
21. }
23. **public** **void** shutdown() {
24. **if** (consumer != **null**) consumer.shutdown();
25. **if** (executor != **null**) executor.shutdown();
26. }
28. **public** **void** run(**int** a\_numThreads) {
29. Map<String, Integer> topicCountMap = **new** HashMap<String, Integer>();
30. topicCountMap.put(topic, **new** Integer(a\_numThreads));
31. Map<String, List<KafkaStream<**byte**[], **byte**[]>>> consumerMap = consumer.createMessageStreams(topicCountMap);
32. List<KafkaStream<**byte**[], **byte**[]>> streams = consumerMap.get(topic);  //得到主题消费流
34. // 启动所有线程
35. executor = Executors.newFixedThreadPool(a\_numThreads);
37. // 开始消费消息
38. **int** threadNumber = 0;
39. **for** (**final** KafkaStream stream : streams) {
40. executor.submit(**new** ConsumerTest(stream, threadNumber));
41. threadNumber++;
42. }
43. }
45. **private** **static** ConsumerConfig createConsumerConfig(String a\_zookeeper, String a\_groupId) {  //消费者配置
46. Properties props = **new** Properties();
47. props.put("zookeeper.connect", "192.168.2.225:2183/config/mobile/mq/mafka");
48. props.put("group.id", "push-token");
49. props.put("zookeeper.session.timeout.ms", "60000");
50. props.put("zookeeper.sync.time.ms", "2000");
51. props.put("auto.commit.interval.ms", "1000");
53. **return** **new** ConsumerConfig(props);
54. }
56. **public** **static** **void** main(String[] args) {
57. String zooKeeper = args[0];
58. String groupId = args[1];
59. String topic = args[2];
60. **int** threads = Integer.parseInt(args[3]);
62. ConsumerGroupExample example = **new** ConsumerGroupExample(zooKeeper, groupId, topic);
63. example.run(threads);
65. **try** {
66. Thread.sleep(10000);
67. } **catch** (InterruptedException ie) {
69. }
70. example.shutdown();
71. }
72. }

总结：

**kafka消费者api分为high api和low api，目前上述demo是都是使用kafka high api，高级api不用关心维护消费状态信息和负载均衡，系统会根据配置参数，**

**定期flush offset到zk上，如果有多个consumer且每个consumer创建了多个线程，高级api会根据zk上注册consumer信息，进行自动负载均衡操作。**

注意事项：

1.高级api将会内部实现持久化每个分区最后读到的消息的offset，数据保存在zookeeper中的消费组名中(如/consumers/push-token-group/offsets/push-token/2。

其中push-token-group是消费组，push-token是topic，最后一个2表示第3个分区)，每间隔一个(默认1000ms)时间更新一次offset，

那么可能在重启消费者时拿到重复的消息。此外，当分区leader发生变更时也可能拿到重复的消息。因此在关闭消费者时最好等待一定时间（10s）然后再shutdown()

2.消费组名是一个全局的信息，要注意在新的消费者启动之前旧的消费者要关闭。如果新的进程启动并且消费组名相同，kafka会添加这个进程到可用消费线程组中用来消费

topic和触发重新分配负载均衡，那么同一个分区的消息就有可能发送到不同的进程中。

3.如果消费者组中所有consumer的总线程数量大于分区数，一部分线程或某些consumer可能无法读取消息或处于空闲状态。

4.如果分区数多于线程数(如果消费组中运行者多个消费者，则线程数为消费者组内所有消费者线程总和)，一部分线程会读取到多个分区的消息

5.如果一个线程消费多个分区消息，那么接收到的消息是不能保证顺序的。

备注：可用[zookeeper web ui工具](http://blog.csdn.net/lizhitao/article/details/30989275)管理查看zk目录树数据： xxx/consumers/push-token-group/owners/push-token/2其中

push-token-group为消费组，push-token为topic,2为分区3.查看里面的内容如：

push-token-group-mobile-platform03-1405157976163-7ab14bd1-0表示该分区被该标示的线程所执行。

总结：

producer性能优化:异步化，消息批量发送，具体浏览上述参数说明。

consumer性能优化:如果是高吞吐量数据，设置每次拿取消息(fetch.min.bytes)大些，拿取消息频繁(fetch.wait.max.ms)些(或时间间隔短些)，如果是低延时要求，则设置时间时间间隔小，每次从kafka broker拿取消息尽量小些。

请注明转载自:http://blog.csdn.net/lizhitao/article/details/37811291