Kafka

### kafka是什么？kafka仅仅是属于消息中间件吗？

kafka在设计之初的时候，开发人员们在除了消息中间件以外，还想把kafka设计为一个能够存储数据的系统，有点像常见的非关系型数据库，比如说NoSql等。除此之外，还希望kafka能支持持续变化，不断增长的数据流, 可以发布和订阅数据流，还可以对于这些数据进行保存。

也就是说kafka的本质是一个数据存储平台，流平台，只是他在做消息发布，消息消费的时候我们可以把他当做消息中间件来用。

而且kafka在设计之初就是采用分布式架构设计的， 基于集群的方式工作，且可以自由伸缩，所以kafka构建集群非常简单！

### 基本概念：

Broker : 和AMQP里协议的概念一样， 就是消息中间件所在的服务器

Topic(主题) : 每条发布到Kafka集群的消息都有一个类别，这个类别被称为Topic。（物理上不同Topic的消息分开存储，逻辑上一个Topic的消息虽然保存于一个或多个broker上但用户只需指定消息的Topic即可生产或消费数据而不必关心数据存于何处）

Partition(分区) : Partition是物理上的概念，体现在磁盘上面，每个Topic包含一个或多个Partition.

Producer : 负责发布消息到Kafka broker

Consumer : 消息消费者，向Kafka broker读取消息的客户端。

Consumer Group（消费者群组） : 每个Consumer属于一个特定的Consumer Group（可为每个Consumer指定group name，若不指定group name则属于默认的group）。

offset 偏移量： 是kafka用来确定消息是否被消费过的标识，在kafka内部体现就是一个递增的数字

kafka消息发送的时候，考虑到性能可以采用打包方式发送，也就是说传统的消息是一条一条发送，现在可以先把需要发送的消息缓存在客户端，等到达一定数值时， 再一起**打包**发送，而且还可以对发送的数据进行**压缩**处理，减少在数据传输时的开销

### kafka优缺点

优点: 基于磁盘的数据存储、高伸缩性、高性能

应用场景: 收集指标和日志、提交日志、流处理

缺点：运维难度大、偶尔有数据混乱的情况、对zookeeper强依赖、多副本模式下对带宽有一定要求

### kafka安装&启动

kafka安装的话，直接从官网下载压缩包下来解压就可以了

注意的是，启动kafka要先启动zookeeper，kafka默认自带了zookeeper 可以启动他自带的，也可以自己另外使用

启动kafka需要执行kafka-server-start.bat 文件，然后需要传入一个路径参数，就是你server.config文件的地址，一般情况下传入../../config/server.properties 即可

踩的坑：因为kafka需要JDK支持，如果JDK安装路径含有空格，需要在kafka-run-class.bat 脚本文件179行，将

set COMMAND=%JAVA% %KAFKA\_HEAP\_OPTS% %KAFKA\_JVM\_PERFORMANCE\_OPTS% %KAFKA\_JMX\_OPTS% %KAFKA\_LOG4J\_OPTS% -cp "%CLASSPATH%" %KAFKA\_OPTS% %\*

将  %CLASSPATH%  改成  "%CLASSPATH%" ，加“”号；

kafka安装路径不要含有空格，不然也不行！

刚刚提到的zookeeper，kafka默认的zookeeper启动的话启动zookeeper-server-start.bat文件即可，同样要传入路径参数：../../config/zookeeper.properties

### server 参数解释：

log.dirs: 日志文件存储地址，可以设置多个，已逗号隔开

num.recovery.threads.per.data.dir：用来读取日志文件的线程数量，对应每一个log.dirs，若此参数设置为2而log.dirs 为2个目录，那么就会有4个线程来读取

auto.create.topics.enable:是否自动创建tiopic

num.partitions: 创建topic的时候自动创建多少个分区 (可以在创建topic的时候手动指定)

log.retention.hours: 日志文件保留时间，超时即删除

log.retention.bytes: 日志文件最大大小

log.segment.bytes: 当日志文件达到一定大小时，开辟新的文件来存储(分片存储)

log.segment.ms: 同上，只是当达到一定时间时，开辟新的文件

message.max.bytes: broker能接收的最大消息大小(单条) 默认1M

### kafka基本管理操作命令:

##列出所有主题

kafka-topics.bat --zookeeper localhost:2181/kafka --list

##列出所有主题的详细信息

kafka-topics.bat --zookeeper localhost:2181/kafka --describe

##创建主题，主题名 my-topic，1副本，8分区

kafka-topics.bat --zookeeper localhost:2181/kafka --create --replication-factor 1 --partitions 8 --topic my-topic

##增加分区，注意：分区无法被删除

kafka-topics.bat --zookeeper localhost:2181/kafka --alter --topic my-topic --partitions 16

##删除主题

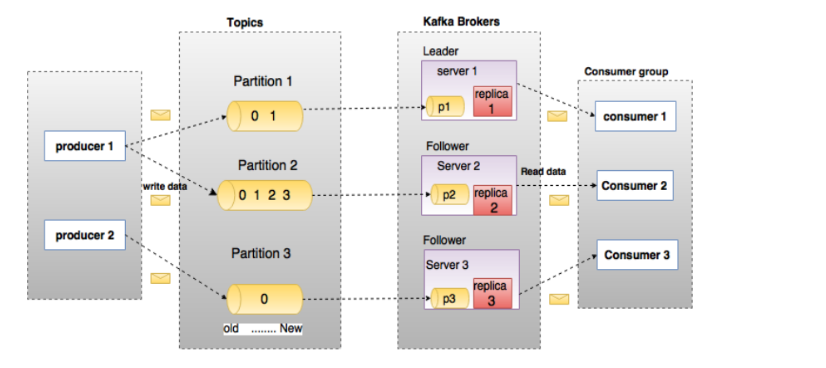
kafka-topics.bat --zookeeper localhost:2181/kafka --delete --topic my-topic

##列出消费者群组（仅Linux）

kafka-topics.sh --new-consumer --bootstrap-server localhost:9092/kafka --list

##列出消费者群组详细信息（仅Linux）

kafka-topics.sh --new-consumer --bootstrap-server localhost:9092/kafka --describe --group 群组名



### kafka java客户端实战

引入maven依赖：

<dependency>  
 <groupId>org.apache.kafka</groupId>  
 <artifactId>kafka-clients</artifactId>  
 <version>0.11.0.0</version>  
</dependency>

注意：如果没创建主题需要先创建后再执行代码

生产者:

public class TestProducter {  
 public static void main(String[] args) throws Exception{  
 Properties properties = new Properties();  
 //指定kafka服务器地址，如果是集群可以指定多个，但是就算只指定一个他也会去集群环境下寻找其他的节点地址  
 properties.setProperty("bootstrap.servers","127.0.0.1:9092");  
 //key序列化器  
 properties.setProperty("key.serializer", StringSerializer.class.getName());  
 //value序列化器  
 properties.setProperty("value.serializer",StringSerializer.class.getName());  
 KafkaProducer<String,String> kafkaProducer = new KafkaProducer<String, String>(properties);  
 ProducerRecord<String, String> stringStringProducerRecord = new ProducerRecord<String, String>("test-topic",1,"testKey","hello");  
 Future<RecordMetadata> send = kafkaProducer.send(stringStringProducerRecord);  
 RecordMetadata recordMetadata = send.get();  
 System.out.println(recordMetadata);  
 }  
}

消费者:

public class TestCousmer {  
 public static void main(String[] args) {  
 Properties properties = new Properties();  
 properties.setProperty("bootstrap.servers","127.0.0.1:9092");  
 properties.setProperty("key.deserializer", StringDeserializer.class.getName()); properties.setProperty("value.deserializer",StringDeserializer.class.getName());  
 properties.setProperty("group.id","1111");  
 KafkaConsumer<String,String> consumer = new KafkaConsumer<String, String>(properties);  
 consumer.subscribe(Collections.singletonList("test-topic"));  
 while (true){  
 ConsumerRecords<String, String> poll = consumer.poll(500);  
 for (ConsumerRecord<String, String> stringStringConsumerRecord : poll) {  
 System.out.println(stringStringConsumerRecord);  
 }  
 }  
 }  
}

### kafka生产者参数详解

acks: 至少要多少个分区副本接收到了消息返回确认消息，一般是0即只要消息发送出去了就确认(不管消息发送是否失败) 1:只要有一个broker接收到了消息就返回，all：所有集群副本都接收到了消息确认，当然 2 3 4 5 这种数字都可以， 就是具体多少台机器接收到了消息返回，但是一般这种情况很少用到

buffer.memory：生产者缓存在本地的消息大小，如果生产者在生产消息的速度过快，快过了往broker发送消息的速度，那么就会出现buffer.memory不足的问题，默认值为32M。注意：单位是byte大概3355000左右

max.block.ms: 生产者获取kafka元数据(集群数据，服务器数据等) 等待时间，当因网络原因导致客户端与服务器通讯时等待的时间超过此值时 会抛出一个TimeOutExctption，默认值为60000ms

retries：设置生产者生产消息失败后重试的次数，默认值 3次

retry.backoff.ms: 设置生产者每次重试的间隔，默认100ms

batch.size: 生产者批次发送消息的大小，默认16k，注意单位还是byte

linger.ms: 生产者生产消息后等待多少毫秒发送到broker，与batch.size 谁先到达就根据谁，默认值为0

compression.type：kafka在压缩数据时使用的压缩算法，可选参数有:none、gzip、snappy，none即不压缩；gzip,和snappy压缩算法之间取舍的话，gzip压缩率比较高，系统cpu占用比较大，但是带来的好处是网络带宽占用少， snappy压缩比没有gzip高，cpu占用率不是很高，性能也还行，如果网络带宽比较紧张的话可以选择gzip，一般推荐snappy

client.id: 一个标识， 可以用来标识消息来自哪，不影响kafka消息生产

max.in.flight.requests.per.connection：指定kafka一次发送请求在得到服务器回应之前,可发送的消息数量，通常与kafka服务端单次接收到的消息大小设置一致，与消费端能接收的一次消息大小也设置成一致。