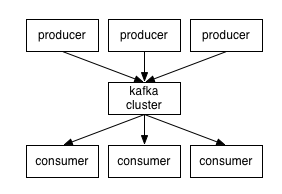
参考：<http://blog.csdn.net/likewindy/article/details/51740837>

**初识Apache Kafka+JAVA程序实例**

　　本文是从英文的官网摘了翻译的，用作自己的整理和记录。水平有限，欢迎指正。版本是： kafka\_2.10-0.10.0.0 

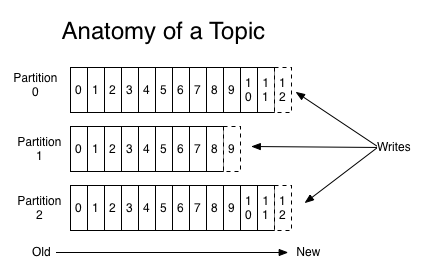
一、基础概念

* 主题：Kafka maintains feeds of messages in categories called topics.
* 生产者：We’ll call processes that publish messages to a Kafka topic producers.
* 消费者：We’ll call processes that subscribe to topics and process the feed of published messages consumers.
* 代理(Broker):Kafka is run as a cluster comprised of one or more servers each of which is called a broker.

　生产者通过网路将消息发送到Kafka集群上，集群依次（轮流）服务消息到达消费者。 Kafka运行在一个集群中，集群中的每一个服务器就叫代理。   


* Partition：Partition 是物理上的概念，每个 Topic 包含一个或多个 Partition，每个parition分为多个副本，副本分布子在不同的机器上，有效的实现负载，每个partition只有一个leader，其他的都为follower。

主题和日志

　　一个主题是命名或分类发布的消息。每一个主题，Kafka持有一个分区日志，看起来像下面图片。　   
　　   
　　每一个Partition都是有序(保存了offset)的，固定长度的消息队列一直不断增加到–一个提交日志。消息在Partition内分配了顺序的id叫偏移量，这个偏移量在分区中唯一标识每个消息的。   
　　Kafka保存所有(一段时间内的-可配置)已经发布的消息(me:非内存存储，而是直接写到了磁盘，充分利用了顺序读写磁盘的高效率)－无论它们是否已经被消费。例如，如果日志保留被设置为两天，那么在一个消息发布后，两天内它是可用的，两天后它将被丢弃到空闲空间

事实上，元数据保留在每个消费进程中，是基于消费进程在日志中的位置，该位置称为“偏移量”（In fact the only metadata retained on a per-consumer basis is the position of the consumer in the log, called the “offset”.）。这个偏移量被消费者控制：正常的消费者读取消息时，线性增加偏移量，但事实上消费者可以以任何它顺序的方式来控制。例如：一个消费者可以重置到以前的偏移量位置来重新处理。   
　　这种组合的特点意味着Kafka的消费者是很廉价的－消费者进程可以随时增加减少，对集群和其它消费者进程没有任何影响。例如：你可以使用命令行工具输出任何主题的内容，而不改变任何现有的消费者所消耗的。   
　　日志中的分区服务几个目的。首先，日志的规模大小可以调整，远不是只有一个在一个服务器上。每个单独的分区都必须安装在主机上的服务器上，一个主题可以有许多分区，所以它可以处理任意数量的数据。第二，它们都是独立相互平行的。

Distribution（分布）

日志的分区分布在Kafka集群中的服务器上，每个服务器处理数据，并请求分区内容的副本。为了容错，每个分区的副本数量是可以通过服务器设置的。   
　   
每个分区都有一个服务器它充当“leader”和0到更多的服务器，作为“followers”。leader处理所有的读写请求，而followers被动地复制leader。如果leader失败，其中一个“followers”将自动成为新的“leader”。 

Producers

生产者将数据发布到他们所选择的主题。生产者负责选择那个消息分配到那个主题的哪个partition。至于选择哪个分区可以简单的循环方式达到负载均衡，也可以者根据语义功能来分区。

Consumers

　　每个消费者把自己标示到一个消费组，当每个消息发布到主题后，消息在投递到每个订阅消费组一个消费实例。消费者实例可以在不同的进程或不同的机器上。   
　　如果所有的消费者实例都有相同的消费组，那么这就像一个传统的队列(me:这时即类似为点对点模式)。   
　　如果所有的消费者实例都有不同的消费组，那么这类作品就如发布订阅，所有的信息都被广播给所有的消费者。   
　　然而，更常见的是主题有一个小数量的消费组，每一个为“逻辑订阅。每个组都是由许多消费实例，为了可扩展性和容错性。   
　　Kafka有比传统消息系统更强壮的顺序保证。   
　　传统的队列在服务器上保留顺序消息，如果多个消费者从队列中消费，然后服务器将它们存储的消息按照顺序发送出去。然而，虽然服务器按照顺序发送消息，但是消息传递异步发送给消费者，所以消息到达消费者时可能失序了。这种高效意味着在并行消费过程中，消息的顺序丢失。消息传递系统经常围绕这个工作，有一个“exclusive consumer“的概念，它只允许一个进程从一个队列中消耗，但当然这意味着没有并行性处理的可能性。   
　　Kafka做得更好。通过对主题进行分区，Kafka是既能保证顺序，又能负载均衡的消费。这是通过给主题进行分区，然后给消费组，使得每个分区都被组内唯一消费进程消费。通过这样做，我们确保消进程是唯一的读取那个分区，并消费数据的顺序。请注意，在一个消费组中，不能有比分区更多的消费进程（me：否则某些消费者将会处于空闲状态）。

Kafka只在一个分区中的消息提供了一个总的顺序，而不是在一个主题中的不同分区之间的。然而，如果您需要一个完全有序的消息，这可以通过一个主题和一个分区来实现，显然这将意味着每一个消费组只有一个消费进程。   
一个主题🡪多个分区🡪一个消费组

一个分区🡪多个消费者

Guarantees（保证）

Kafka给出了以下保证：

* 生产者发送到一个特定主题的分区的消息，将被添加，并且发送是顺序的。
* 各消费实例看到消息是顺序，并且存储在日志里而不是内存中。
* 一个主题由Ｎ各复制备份（实际是分区的备份），我们将容忍Ｎ-1服务器故障而不丢失任何信息提交到日志。

二、程序实例

重要的来了，上面看不懂的没关系，看程序，最直接。   
假如我们有一个主题叫foo,它有４个分区。我建立了两个消费组GroupA and GroupB   
　   
其中GroupA有２个消费者，GroupB有４个消费者。   
我们的生产者平均向４个分区写入了内容。例：

package part;

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 TestProducer {

public static void main(String[] args) {

Properties props = new Properties();//配置kafka

props.put("bootstrap.servers", "localhost:9092");

//The "all" setting we have specified will result in blocking on the full commit of the record, the slowest but most durable setting.

//“所有”设置将导致记录的完整提交阻塞，最慢的，但最持久的设置。

props.put("acks", "all");

//如果请求失败，生产者也会自动重试，即使设置成０ the producer can automatically retry.

props.put("retries", 0);

//The producer maintains buffers of unsent records for each partition.

props.put("batch.size", 16384);

//默认立即发送，这里这是延时毫秒数

props.put("linger.ms", 1);

//生产者缓冲大小，当缓冲区耗尽后，额外的发送调用将被阻塞。时间超过max.block.ms将抛出TimeoutException

props.put("buffer.memory", 33554432);

//The key.serializer and value.serializer instruct how to turn the key and value objects the user provides with their ProducerRecord into bytes.

props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");

props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

//创建kafka的生产者类

Producer<String, String> producer = new KafkaProducer<String, String>(props);

//生产者的主要方法

// close();//Close this producer.

// close(long timeout, TimeUnit timeUnit); //This method waits up to timeout for the producer to complete the sending of all incomplete requests.

// flush() ;所有缓存记录被立刻发送

for(int i = 0; i < 100; i++)

　　　　　　//这里平均写入４个分区

producer.send(new ProducerRecord<String, String>("foo",i%4, Integer.toString(i), Integer.toString(i)));

producer.close();

}

}

消费者

package part;

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 TestConsumer {

public static void main(String[] args) {

Properties props = new Properties();//配置kafka

props.put("bootstrap.servers", "localhost:9092");

System.out.println("this is the group part test 1");

//消费者的组id//设置消费者组

props.put("group.id", "GroupA");//这里是GroupA或者GroupB

props.put("enable.auto.commit", "true");

props.put("auto.commit.interval.ms", "1000");

//从poll(拉)的回话处理时长

props.put("session.timeout.ms", "30000");

//poll的数量限制

　　　　　//props.put("max.poll.records", "100");

props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(props);//创建消费者client

//订阅主题列表topic

consumer.subscribe(Arrays.asList("foo"));//以组的形式来订阅主题

while (true) {//循环处理消息

ConsumerRecords<String, String> records = consumer.poll(100);

for (ConsumerRecord<String, String> record : records)

//　正常这里应该使用线程池处理，不应该在这样处理

System.out.printf("offset = %d, key = %s, value = %s", record.offset(), record.key(), record.value()+"\n");

}

}

}

如果GroupA和GroupB都正常启动，那么GroupB内４个消费平均消费生产者的消息数据（这里每个２５个消息），GroupA内２个消费者各处理５０个消息，每个消费者处理2个分区。如果GroupA内一个消费者挂断，那么另一个处理所有消息数据。如果GroupB挂掉一个，那么将有一个消费者出来处理挂掉没有处理的消息数据。   
　　以下命令可以修改某主题的分区大小。

bin/kafka-topics.sh --zookeeper localhost:2181 --alter --topic foo --partitions 4

* 1

三、multi-broker cluster

这里其实和Zookeeper机制由点类似，也是建立了一个leader和几个follower。主要的作用还是为了可扩展性和容错性。当集中任意一台出问题，都可以保证系统的正确和稳定。即使是leader出现问题，它们也可以通过投票的方式产生新leader. 这里只是简单说明一下。

在它的官方例子中通过复制原有的配置文件，在本地建立了伪集群服务。

> cp config/server.properties config/server-1.properties

> cp config/server.properties config/server-2.properties

config/server-1.properties:

broker.id=1

listeners=PLAINTEXT://:9093

log.dir=/tmp/kafka-logs-1

config/server-2.properties:

broker.id=2

listeners=PLAINTEXT://:9094

log.dir=/tmp/kafka-logs-2

其中 broker.id 属性是集群中唯一的和永久的节点名字，正常应该是一台机子一个服务。其它两个是因为伪集群的原因必须修改。   
让后启动这两台服务建立伪集群。模拟了leader失效（被强行kill）后，它还可以正常工作。   
启动：

> bin/kafka-server-start.sh config/server-1.properties &

...

> bin/kafka-server-start.sh config/server-2.properties &

四、典型应用场景

1. 监控/日志：主机通过Kafka发送与系统和应用程序健康相关的指标，然后这些信息会被收集和处理从而创建监控仪表盘并发送警告。除此之外，LinkedIn还利用Apache Samza实现了一个能够实时处理事件的富调用图分析系统。
2. 传统的消息： 应用程度使用Kafka作为传统的消息系统实现标准的队列和消息的发布—订阅（redis内置了发布订阅机制），例如搜索和内容提要（Content Feed）。
3. 分析: 为了更好地理解用户行为，改善用户体验，LinkedIn会将用户查看了哪个页面、点击了哪些内容等信息发送到每个数据中心的Kafka集群上，并通过Hadoop进行分析、生成日常报告。
4. 作为分布式应用程序或平台的构件（日志）：大数据仓库解决方案Pinot等产品将Kafka作为核心构件（分布式日志），分布式数据库Espresso将其作为内部副本并改变传播层。

英文原地址：<http://kafka.apache.org/documentation.html#quickstart>