kafka企业级消息系统

课程目标

- 什么是消息系统
- 消息系统应用场景
- kafka基本架构了解
- kafka集群搭建掌握
- kafka基本操作掌握
- kafka核心术语及各个术语之间的关系

运行环境

- linux系统
- jdk安装
- kafka安装包

1、了解为什么要使用消息系统

在没有使用消息系统以前，我们传统对于多业务、夸服务消息传递的时候，会采用串行方式或者并行方法；

- 串行方式：

  用户注册实例：将注册信息写入数据库成功后，发送注册邮件，在发送注册短信。

- 并行方式：

  将注册信息写入数据库成功后，发送注册邮件的同事，发送注册短信。以上三个任务完成之后，发挥给客户端，与串行的差别是并行的方式可以提提交处理的时间。

消息系统：

- 消息系统负责将数据从一个应用程序传送到另一个应用程序，因此应用程序可以专注于数据，但是不必担心如何共享它。分布式消息产地基于可靠的消息队列的概念。消息在客户端应用程序和消息传递系统之间的异步排队。
- 有两种类型的消息模式可用-点对点；发布-订阅消息系统
  - 点对点消息系统中，消息被保留在队列中，一个或者多个消费者可以消耗队列中的消息，但是特定的消息只能有最多的一个消费者消费。一旦丽霞飞着读取队列中的消息，他就从该队列中消失。该系统的典

型梳理就是订单处理系统，其中每个订单将有一个订单处理器处理，但多个订单处理器可以同时工作。

- 大多数的消息系统是基于发布-订阅消息系统

1.1、消息中间件|消息系统

是从一个系统将数据传递给另一个系统；如果单纯只是传递数据的方式有很多：http、rpc、webservice、定时任务。

2、了解消息系统的分类

2.1、点对点

主要采用的队列的方式，如A->B 当B消费的队列中的数据，那么队列的数据就会被删除掉【如果B椅子不消费那么久会存在队列中有很多的脏数据】

2.2、发布-订阅

必须要有主题的概念；主题：一个消息的分类

发布者将消息财通推方式给消息系统；订阅者可以采用拉、推的方式从消息系统中拿数据

3、消息系统的应用场景

3.1、应用解耦

将一个大型的任务系统分成若干个小模块，将所有的消息进行统一的管理和存储，因此为了解耦，就会涉及到kafka企业级消息平台

3.2、流量控制

秒杀活动当中，一般会因为流量过大，应用服务器挂掉，为了解决这个问题，一般需要在应用前端假如消息队列。

- 可以控制活动的人数

- 可以缓解短时间内流量大使服务崩掉。

3.3、日志处理

- 日志处理指将消息队列用在日志处理中，比如kafka的应用中，解决大量的日志传输问题；日志采集工具采集数据写入kafka中；kafka消息队列负责日志数据的接收，存储，转发功能；日志处理应用程序：订阅并消费kafka队列中的数据，进行数据分析。

3.4、消息通讯

- 消息队列一般都内置了高校的通信机制，因此也可以用在纯的消息通讯，比如点对点的消息队列，或者聊天室等。

4、kafka简介

4.1、简介

kafka是最初由linkedin公司开发的，使用scala语言编写，kafka是一个分布式，分区的，多副本的，多订阅者的日志系统（分布式MQ系统），可以用于搜索日志，监控日志，访问日志等。

4.2、支持的语言

kafka目前支持多种客户端的语言：java、python、c++、php等

4.3、apache kafka是一个分布式发布-订阅消息系统

apache kafka是一个分布式发布-订阅消息系统和一个强大的队列，可以处理大量的数据，并使能够将消息从一个端点传递到另一个端点，kafka适合离线和在线消息消费。kafka消息保留在磁盘上，并在集群内复制以防止数据丢失。kafka构建在zookeeper同步服务之上。它与apache和spark非常好的继承，应用 于实时流式数据分析。

4.4、其他的消息队列

RabbitMQ

Redis

ZeroMQ

ActiveMQ

4.5、kafka的好处

可靠性：分布式的，分区，复制和容错的。

可扩展性：kafka消息传递系统轻松缩放，无需停机。

耐用性：kafka使用分布式提交日志，这意味着消息会尽可能快速的保存在磁盘上，因此它是持久的。

性能：kafka对于发布和定于消息都具有高吞吐量。及时存储了许多TB的消息，他也爆出稳定的新能。

kafka非常快：保证零停机和零数据丢失。

5、kafka应用场景

5.1、指标分析

kafka 通常用于操作监控数据。这设计聚合来自分布式应用程序的统计信息， 以产生操作的数据集中反馈

5.2、日志聚合解决方法

kafka可用于跨组织从多个服务器手机日志，并使他们以标准的合适提供给多个服务器。

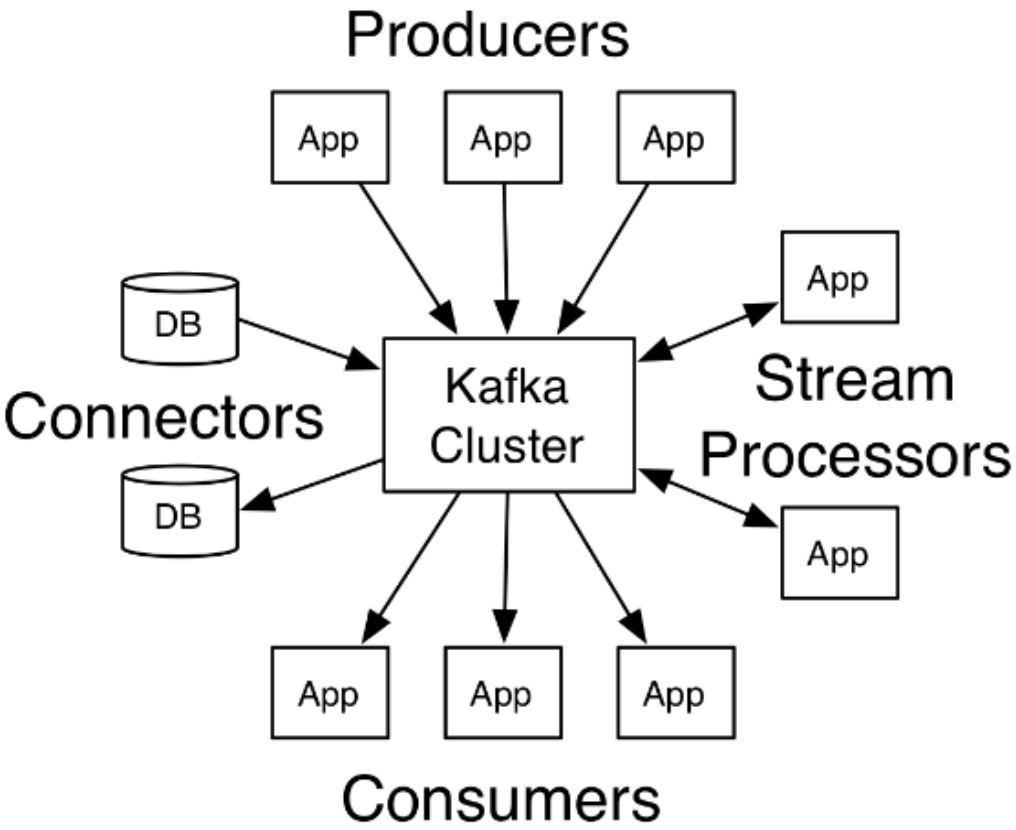
5.3、流式处理

流式处理框架（spark，storm，flink）重主题中读取数据，对齐进行处理，并将处理后的数据写入新的主题，供用户和应用程序使用，kafka的强耐久性在流处理的上下文中也非常的有用。

6、kafka架构

6.1、kafka的架构图

6.1.1、官方文档架构图：



6.1.1.1、kafka四大核心

6.1.1.1.1、生产者API

允许应用程序发布记录流至一个或者多个kafka的主题（topics）。

6.1.1.1.2、消费者API

允许应用程序订阅一个或者多个主题，并处理这些主题接收到的记录流。

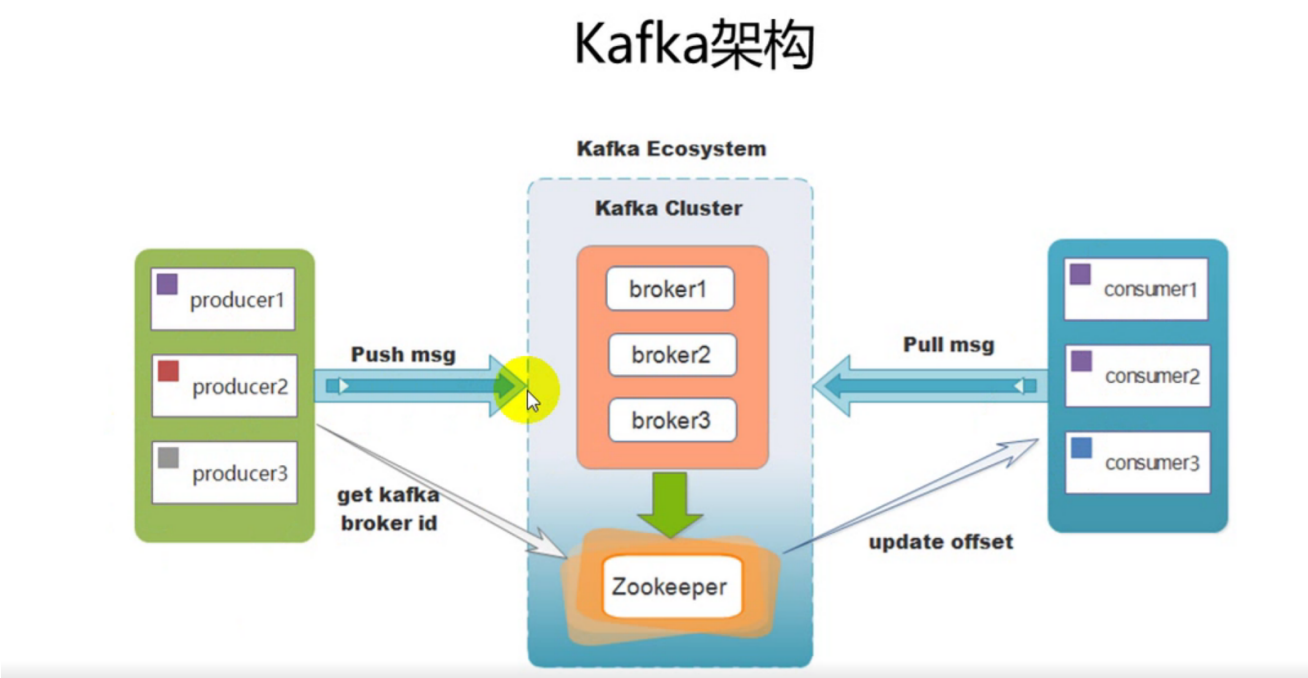
6.1.1.1.3、StreamsAPI

允许应用程序充当流处理器（stream processor），从一个或者多个主题获取输入流，并生产一个输出流到一个或者多个主题，能够有效的变化输入流为输出流。

6.1.1.1.4、ConnectorAPI

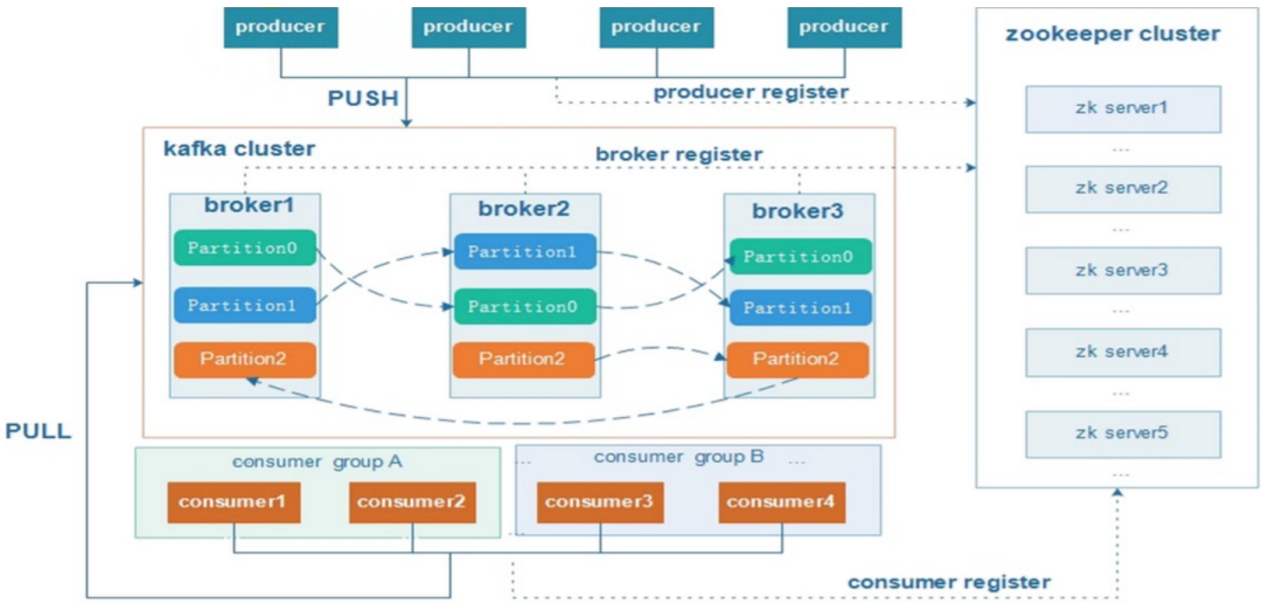
允许构建和运行可重用的生产者或者消费者，能够把kafka主题连接到现有的应用程序或数据系统。例如：一个连接到关系数据库的连接器可能会获取每个表的变化。

6.1.2、架构关系图：



说明：kafka支持消息持久化，消费端为拉模型来拉取数据，消费状态和订阅关系有客户端负责维护，消息消费完后，不会立即删除，会保留历史消息。因此支持多订阅时，消息只会存储一份就可以了。

6.1.3、kafka整体架构图



6.1.4、kafka架构说明

一个典型的kafka集群中包含若干个Producer，若干个Broker，若干个Consumer，以及一个zookeeper集群；kafka通过zookeeper管理集群配置，选举leader，以及在Consumer Group发生变化时进行Rebalance（负载均衡）；Producer使用push模式将消息发布到Broker；Consumer使用pull模式从Broker中订阅并消费消息。

7、kafka术语

7.1、kafka中术语介绍

Broker：kafka集群中包含一个或者多个服务实例，这种服务实例被称为Broker

Topic：每条发布到kafka集群的消息都有一个类别，这个类别就叫做Topic

Partition：Partition是一个物理上的概念，每个Topic包含一个或者多个PatITion

Producer：负责发布消息到kafka的Broker中。

Consumer：消息消费者,向kafka的broker中读取消息的客户端

Consumer Group：每一个Consumer属于一个特定的Consumer Group（可以为每个Consumer指定groupName）

7.2、kafka中topic说明

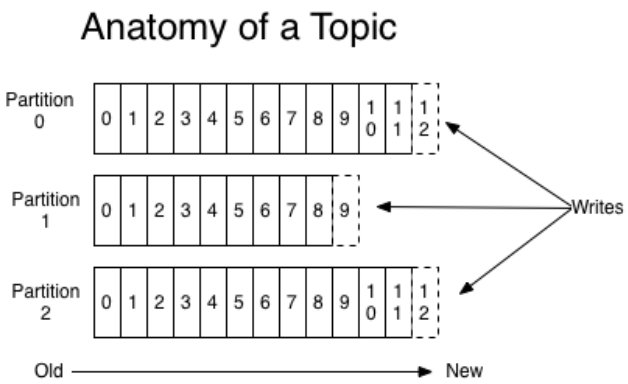
- kafka将消息以topic为单位进行归类
- topic特指kafka处理的消息源（feeds of messages）的不同分类。
- topic是一种分类或者发布的一些列记录的名义上的名字。kafka主题始终是支持多用户订阅的；也就是说，一个主题可以有零个，一个或者多个消费者订阅写入的数据。
- 在kafka集群中，可以有无数的主题。
- 生产者和消费者消费数据一般以主题为单位。更细粒度可以到分区级别。

7.3、kafka中分区数（Partitions）

- **Partitions：分区数**

```
Partitions：分区数：控制topic将分片成多少个log，可以显示指定，如果不指定则会使用
broker（server.properties）中的num.partitions配置的数量。
```

- 一个broker服务下，是否可以创建多个分区？
  - 可以的，broker数与分区数没有关系；
- 在kafka中，每一个分区会有一个编号：编号从0开始
- 某一个分区的数据是有序的



- 说明-数据是有序

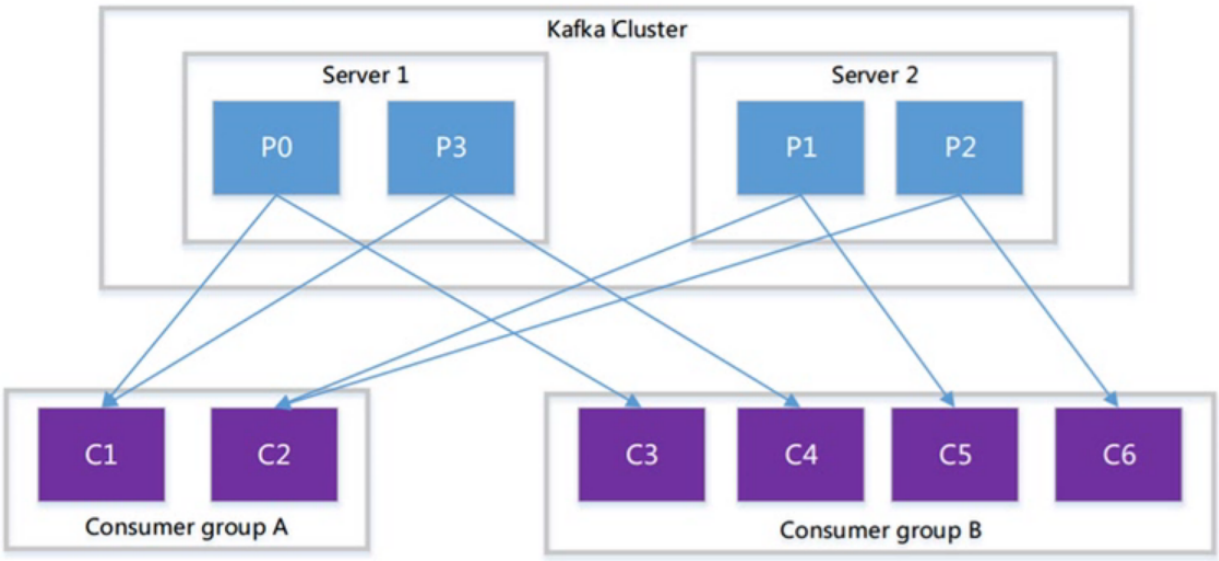
  如何保证一个主题下的数据是有序的？（生产是什么样的顺序，那么消费的时候也是什么样的顺序）

  ```
  一个主题（topic）下面有一个分区（）即可
  ```

- topic的Partition数量在创建topic时配置。

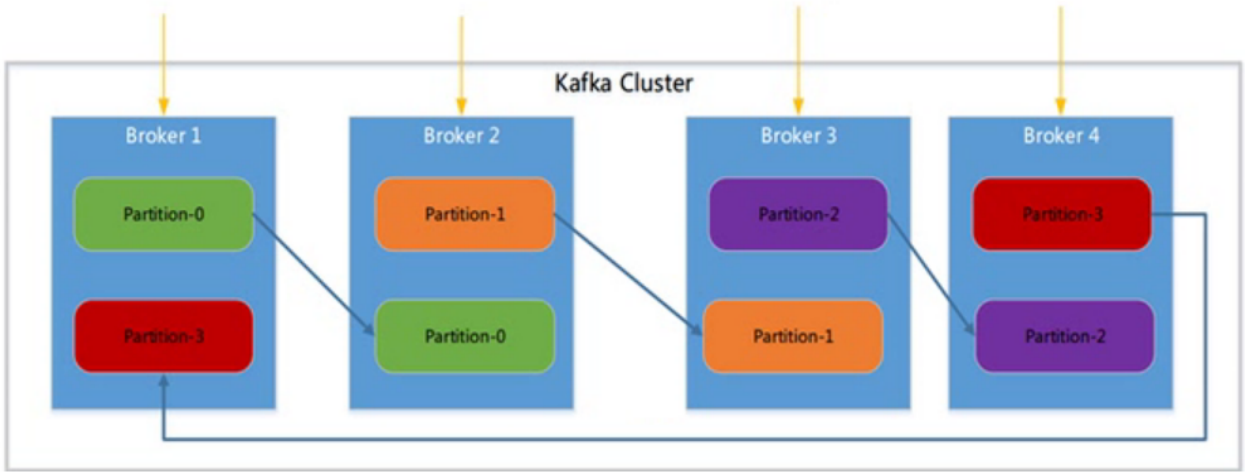
- Partition数量决定了每个Consumer group中并发消费者的最大数量。

- Consumer group A 有两个消费者来读取4个partition中数据；Consumer group B有四个消费者来读取4个partition中的数据



7.4、kafka中副本数（Partition Replication）

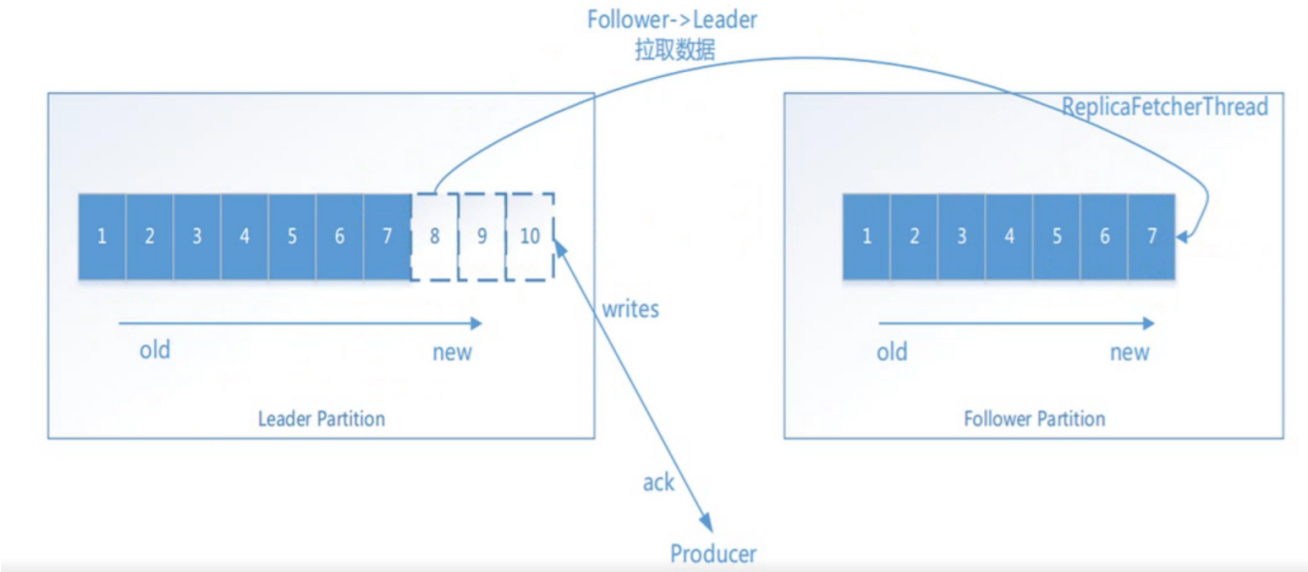
- kafka分区副本数（kafka Partition Replicas)



- 副本数（replication-factor）

> 副本数（replication-factor）：控制消息保存在几个broker（服务器）上，一般情况下等于broker的个数。

- 一个broker服务下，是否可以创建多个副本因子?
  - 不可以；创建主题时，副本因子应该小于等于可用的broker数。
- 副本因子过程图



- 副本因子操作以分区为单位的。每个分区都有各自的主副本和从副本；主副本叫做leader，从副本叫做follower（在有多个副本的情况下，kafka会为同一个分区下的分区，设定角色关系：一个leader和N个follower），处于同步状态的副本叫做**in-sync-replicas**(ISR);follower通过拉的方式从leader同步数据。消费者和生产者都是从leader读写数据，不与follower交互。

- **副本因子的作用**：让kafka读取数据和写入数据时的可靠性。

- 副本因子是包含本身|同一个副本因子不能放在同一个Broker中。

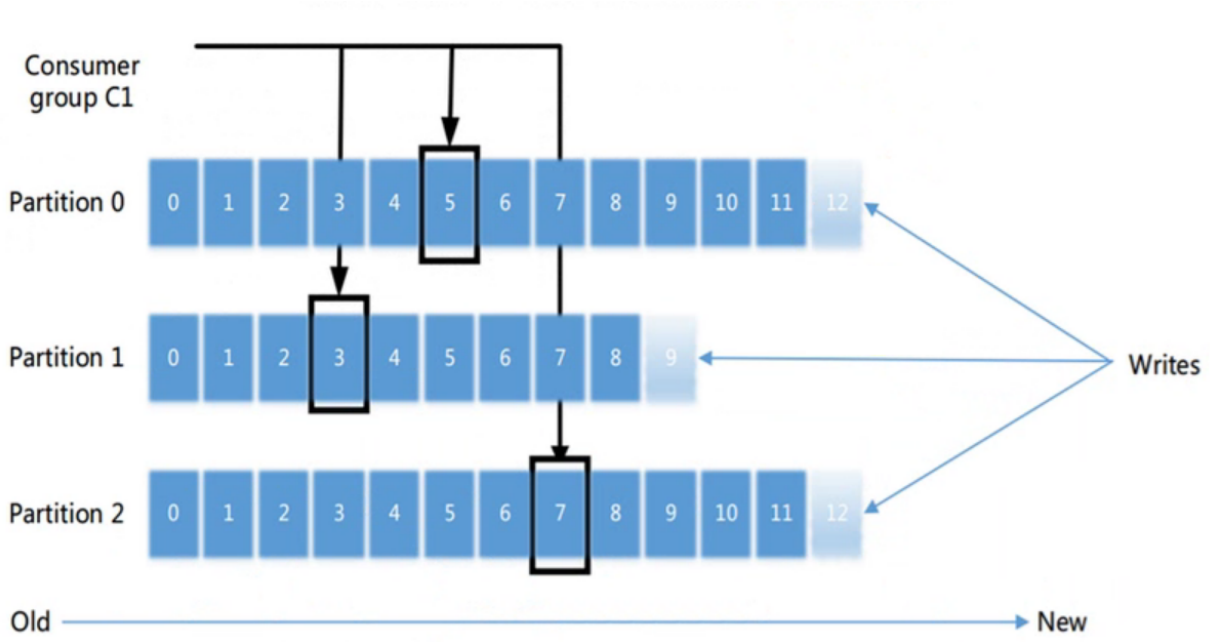
- 如果某一个分区有三个副本因子，就算其中一个挂掉，那么只会剩下的两个钟，选择一个leader，但不会在其他的broker中，另启动一个副本（因为在另一台启动的话，存在数据传递，只要在机器之间有数据传递，就会长时间占用网络IO，kafka是一个高吞吐量的消息系统，这个情况不允许发生）所以不会在零个broker中启动。

- 如果所有的副本都挂了，生产者如果生产数据到指定分区的话，将写入不成功。

- lsr表示：当前可用的副本。

7.5、kafka Partition offset

任何发布到此partition的消息都会被直接追加到log文件的尾部，每条消息在文件中的位置称为offset（偏移量），offset是一个long类型数字，它唯一标识了一条消息，消费者通过（offset，partition，topic）跟踪记录。



7.6、kafka分区和消费组之间的关系

消费组：

由一个或者多个消费者组成，同一个组中的消费者对于同一条消息只消费一次。

某一个主题下的分区数，对于消费组来说，应该小于等于该主题下的分区数。如下所示：

```
如：某一个主题有4个分区，那么消费组中的消费者应该小于4，而且最好与分区数成整数倍
1   2   4
同一个分区下的数据，在同一时刻，不能同一个消费组的不同消费者消费
```

8、kafka集群的搭建

8.1、准备三台机器

```
192.168.140.128 hadoop-01
192.168.140.129 hadoop-02
192.168.140.130 hadoop-03
```

8.2、初始化环境

8.2.1、安装jdk、安装zookeeper

```
略。。。
```

8.2.2、安装目录规划

```
安装包存放的目录：/export/software
安装程序存放的目录：/export/servers
数据目录：/export/data
日志目录：/export/logs


mkdir -p /export/servers
mkdir -p /export/software
mkdir -p /export/data
mkdir -p /export/logs
```

8.2.3、安装用户

如果默认用户安装，即可跳过该步骤

```
安装hadoop，会创建一个hadoop用户
安装kafka，创建一个kafka用户

或者 创建bigdata用户，用来安装所有的大数据软件。

本例：使用root用户
```

8.2.4、验证环境

1、校验java环境

java -verson

```
C:\Users\Deborah>java -version
java version "1.8.0_65"
Java(TM) SE Runtime Environment (build 1.8.0_65-b17)
Java HotSpot(TM) 64-Bit Server VM (build 25.65-b01, mixed mode)
```

2、检验zk环境

zkServer.sh status

```
[root@hadoop-01 ~]# zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /export/servers/zookeeper-3.4.9/bin/../conf/zoo.cfg
Mode:follower
```

3、防火墙关闭

```
[root@hadoop-01 ~]# service iptables status
iptables: Firewall is not running.
[root@hadoop-01 ~]#
```

8.3、kafka集群安装

8.3.1、下载地址

```
https://archive.apache.org/dist/kafka/1.0.0/kafka_2.11-1.0.0.tgz
```

8.3.2、上传到linux服务器

```
rz kafka_2.11-1.0.0.tgz
```

8.3.3、解压

```
tar -zxvf kafka_2.11-1.0.0.tgz -C /export/servers/
```

8.3.4、重命名

```
mv kafka_2.11-1.0.0 kafka
```

8.3.5、配置环境变量

```
#kafka_home
export KAFKA_HOME=/export/servers/kafka
export PATH=$PATH:$KAFKA_HOME/bin
```

8.3.6、让环境变量生效

```
source /etc/profile
```

8.3.7、修改配置文件

vim server.properties

```
vim server.properties
#需要修改以下三个地方

#1) broker.id 标识了kafka集群中一个唯一broker。
broker.id=0

#2) 数据存放的目录，注意目录如果不存在，需要新建下。
#2) 存放生产者生产的数据 数据一般以topic的方式存放
#2) 创建一个数据存放目录 /export/data/kafka  ---  mkdir -p /export/data/kafka
log.dirs=/export/data/kafka

#3) # zk的信息
zookeeper.connect=hadoop-01:2181,hadoop-02:2181,hadoop-03:2181
```

8.3.8、分发安装包

```
scp -r /export/servers/kafka/ hadoop-02:/export/servers/
scp -r /export/servers/kafka/ hadoop-03:/export/servers/
```

8.3.9、修改配置文件

在**hadoop-02**机器上修改server.properties

```
broker.id=1
```

hadoop-02目录数据存放的目录不存在，需要重新创建

```
mkdir -p /export/data/kafka
```

在**hadoop-03**机器上修改server.properties

```
broker.id=2
```

hadoop-03机器上目录数据存放的目录不存在，需要重新创建

```
mkdir -p /export/data/kafka
```

8.4、启动集群

注意事项：在kafka启动前，一定要让zookeeper启动起来。

8.4.1、启动命令-前台启动

```
cd /export/servers/kafka/bin
./kafka-server-start.sh /export/servers/kafka/config/server.properties
```

8.4.2、启动命令-后台启动

```
cd /export/servers/kafka/bin
nohup ./kafka-server-start.sh /export/servers/kafka/config/server.properties &
```

8.4.3、停止命令

```
#1、直接将进程杀掉
jps查看进程，找到kafka的Pid，然后kill掉即可

kill -9 kafkaPID

#2、通过停止脚本停止服务
./kafka-server-stop.sh
```

8.5、查看kafka启动进程

通过jps命令来查看进程是否存在

```
#输入jps

[root@hadoop-01 ~]# jps
2276 QuorumPeerMain
2687 Jps
3056 Kafka
[root@hadoop-01 ~]#
```

9、Kafka集群操作

9.1、kafka集群操作-控制台操作

9.1.1、创建一个Topic

Let's create a topic named "test" with a single partition and only one replica（创建了一个名字为test的主题，有一个分区，有一个副本）

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

9.1.2、查看主题命令

```
bin/kafka-topics.sh --list --zookeeper localhost:2181
```

9.1.3、生产者生产数据

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

9.1.4、消费者消费数据

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from-
beginning
```

9.1.5、创建多副本Topic

create a new topic with a replication factor of three（创建了一个topic，该topic有一个分区，三个副本）

```
bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 3 --
partitions 1 --topic my-replicated-topic
```

9.1.6、运行describe topics命令

```
bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic my-replicated-topic
```

执行完上述的命令后，输出的结果如下所示：

```
Topic:my-replicated-topic   PartitionCount:1    ReplicationFactor:3 Configs:
    Topic: my-replicated-topic  Partition: 0    Leader: 1   Replicas: 1,2,0 Isr: 1,2,0
```

结果说明：

```
        这是输出的解释。第一行给出了所有分区的摘要，每个附加行提供有关一个分区的信息。由于我们只有一个分区用于此主题，因此只有一行。

“leader”是负责给定分区的所有读取和写入的节点。每个节点将成为随机选择的分区部分的领导者。（因为在kafka中如果有多个副本的话，就会存在leader和follower的关系，表示当前这个副本为leader所在的broker是哪一个）
“replicas”是复制此分区日志的节点列表，无论它们是否为领导者，或者即使它们当前处于活动状态。（所有副本列表 0 , 1,2）
“isr”是“同步”复制品的集合。这是副本列表的子集，该列表当前处于活跃状态并且已经被领导者捕获。（可用的列表数）
```

9.1.7、修改topic属性

9.1.7.1、增加topic分区数

```
kafka-topics.sh --zookeeper zkhost:port --alter --topic topicName --partitions 40
```

9.1.7.2、增加配置

```
kafka-topics.sh --zookeeper zkhost:port --alter --topic topicName --config
flush.messages=1
```

9.1.7.3、删除配置

```
kafka-topics.sh --zookeeper zkhost:port --alter --topic topicName --delete-config
flush.messages
```

9.1.7.4、删除topic

目前删除炒作在默认情况下知识打上一个删除的标记，在重新启动kafka后才删除。如果需要立即删除，则需要在server.properties中配置：

```
delete.topic.enable=true
```

```
kafka-topics.sh --zookeeper zkhost:port --delete --topic topicName
```

9.2、kafka集群操作-JavaAPI操作

9.2.1、添加依赖

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>0.11.0.1</version>
</dependency>
```

9.2.2、生产者代码

```
/**
 * 订单的生产者代码
 */
public class OrderProducer {
    public static void main(String[] args) throws InterruptedException {
        /* 1、连接集群，通过配置文件的方式
         * 2、发送数据-topic:order，value
         */
        Properties props = new Properties();
        props.put("bootstrap.servers", "hadoop-01:9092");
        props.put("acks", "all");
        props.put("retries", 0);
        props.put("batch.size", 16384);
        props.put("linger.ms", 1);
```

```
        props.put("buffer.memory", 33554432);
        props.put("key.serializer",
                "org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer",
                "org.apache.kafka.common.serialization.StringSerializer");
        KafkaProducer<String, String> kafkaProducer = new KafkaProducer<String, String>
(props);
        for (int i = 0; i < 1000; i++) {
            // 发送数据 ,需要一个producerRecord对象,最少参数 String topic, V value
            kafkaProducer.send(new ProducerRecord<String, String>("order", "订单信
息! "+i));
            Thread.sleep(100);
        }
    }
}
```

9.2.3、消费者代码

```
/**
 * 消费订单数据--- javaben.tojson
 */
public class OrderConsumer {
    public static void main(String[] args) {
        // 1\连接集群
        Properties props = new Properties();
        props.put("bootstrap.servers", "hadoop-01:9092");
        props.put("group.id", "test");

        //以下两行代码 ---消费者自动提交offset值
        props.put("enable.auto.commit", "true");
        props.put("auto.commit.interval.ms", "1000");


        props.put("key.deserializer",
                "org.apache.kafka.common.serialization.StringDeserializer");
        props.put("value.deserializer",
                "org.apache.kafka.common.serialization.StringDeserializer");
        KafkaConsumer<String, String> kafkaConsumer = new KafkaConsumer<String, String>
(props);
//        2、发送数据 发送数据需要，订阅下要消费的topic。  order
        kafkaConsumer.subscribe(Arrays.asList("order"));
        while (true) {
            ConsumerRecords<String, String> consumerRecords =
kafkaConsumer.poll(100);// jdk queue offer插入、poll获取元素。 blockingqueue put插入原生,
take获取元素
            for (ConsumerRecord<String, String> record : consumerRecords) {
                System.out.println("消费的数据为: " + record.value());
            }
        }
    }
```

```
}
```