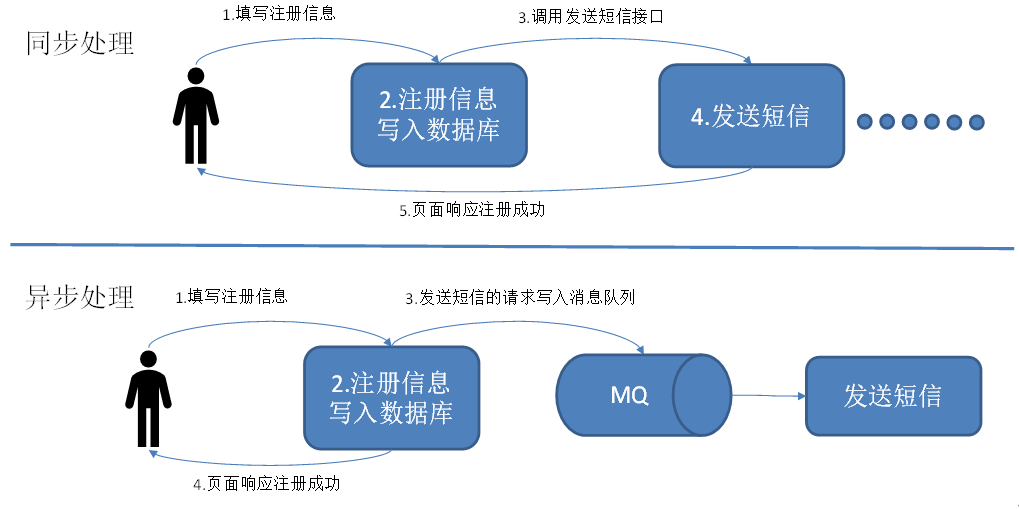
# 第1章 Kafka概述

## 1.1 定义

Kafka是一个分布式的基于发布/订阅模式的消息队列（Message Queue），主要应用于大数据实时处理领域。

## 1.2 消息队列

**1.2.1 传统消息队列的应用场景**



**使用消息队列的好处**

1）解耦

允许你独立的扩展或修改两边的处理过程，只要确保它们遵守同样的接口约束。

2）可恢复性

系统的一部分组件失效时，不会影响到整个系统。消息队列降低了进程间的耦合度，所以即使一个处理消息的进程挂掉，加入队列中的消息仍然可以在系统恢复后被处理。

3）缓冲

有助于控制和优化数据流经过系统的速度，解决生产消息和消费消息的处理速度不一致的情况。

4）灵活性 & 峰值处理能力

在访问量剧增的情况下，应用仍然需要继续发挥作用，但是这样的突发流量并不常见。如果为以能处理这类峰值访问为标准来投入资源随时待命无疑是巨大的浪费。使用消息队列能够使关键组件顶住突发的访问压力，而不会因为突发的超负荷的请求而完全崩溃。

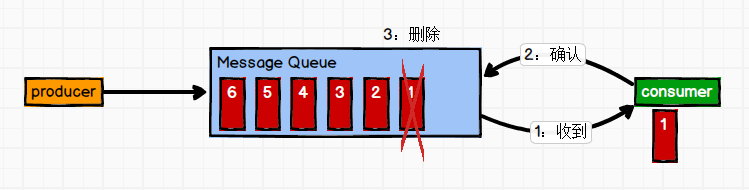
5）异步通信

很多时候，用户不想也不需要立即处理消息。消息队列提供了异步处理机制，允许用户把一个消息放入队列，但并不立即处理它。想向队列中放入多少消息就放多少，然后在需要的时候再去处理它们。

**1.2.2 消息队列的两种模式**

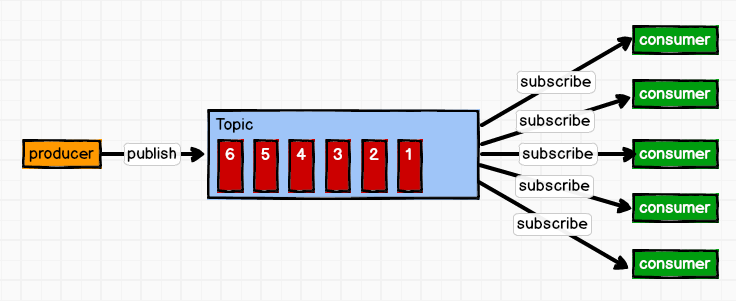
**（1）点对点模式**（一对一，消费者主动拉取数据，消息收到后消息清除）

消息生产者生产消息发送到Queue中，然后消息消费者从Queue中取出并且消费消息。

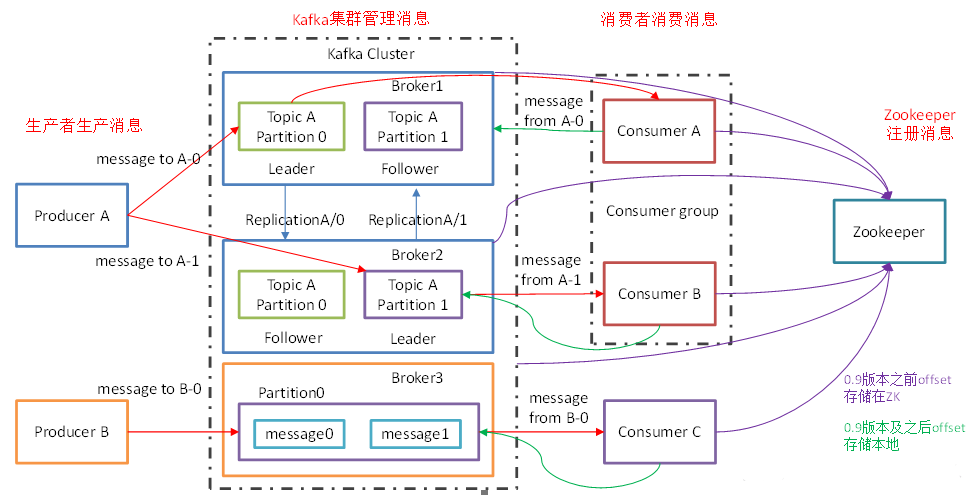
消息被消费以后，queue中不再有存储，所以消息消费者不可能消费到已经被消费的消息。Queue支持存在多个消费者，但是对一个消息而言，只会有一个消费者可以消费。

**（2）发布/订阅模式**（一对多，消费者消费数据之后不会清除消息）

消息生产者（发布）将消息发布到topic中，同时有多个消息消费者（订阅）消费该消息。和点对点方式不同，发布到topic的消息会被所有订阅者消费。



## 1.3 Kafka基础架构



**1）Producer ：**消息生产者，就是向kafka broker发消息的客户端；

**2）Consumer ：**消息消费者，向kafka broker取消息的客户端；

**3）Consumer Group （CG）：**消费者组，由多个consumer组成。**消费者组内每个消费者负责消费不同分区的数据，一个分区只能由一个组内消费者消费；消费者组之间互不影响。**所有的消费者都属于某个消费者组，即**消费者组是逻辑上的一个订阅者**。

**4）Broker ：**一台kafka服务器就是一个broker。一个集群由多个broker组成。一个broker可以容纳多个topic。

**5）Topic ：**可以理解为一个队列，**生产者和消费者面向的都是一个topic**；

**6）Partition：**为了实现扩展性，一个非常大的topic可以分布到多个broker（即服务器）上，**一个topic可以分为多个partition**，每个partition是一个有序的队列；

**7）Replica：**副本，为保证集群中的某个节点发生故障时，该节点上的partition数据不丢失，且kafka仍然能够继续工作，kafka提供了副本机制，一个topic的每个分区都有若干个副本，一个**leader**和若干个**follower**。

**8）leader：**每个分区多个副本的“主”，生产者发送数据的对象，以及消费者消费数据的对象都是leader。

**9）follower：**每个分区多个副本中的“从”，实时从leader中同步数据，保持和leader数据的同步。leader发生故障时，某个follower会成为新的leader。

# 第2章 Kafka快速入门

## 2.1 安装部署

**2.1.1 集群规划**

hadoop102 hadoop103 hadoop104

zk zk zk

kafka kafka kafka

**2.1.2 Kafka 下载**

<http://kafka.apache.org/downloads.html>

**2.1.3 集群部署**

**1）**解压安装包

[user1@hadoop102 software]$ tar -zxvf kafka\_2.11-2.4.1.tgz -C /opt/module/

**2）**修改解压后的文件名称

[user1@hadoop102 module]$ mv kafka\_2.11-2.4.1.tgz kafka

**3）**在/opt/module/kafka目录下创建logs文件夹

[user1@hadoop102 kafka]$ mkdir logs

**4）**修改配置文件

[user1@hadoop102 kafka]$ cd config/

[user1@hadoop102 config]$ vi server.properties

输入以下内容：

#broker的全局唯一编号，不能重复

broker.id=0

#删除topic功能使能,当前版本此配置默认为true，已从配置文件移除

delete.topic.enable=true

#处理网络请求的线程数量

num.network.threads=3

#用来处理磁盘IO的线程数量

num.io.threads=8

#发送套接字的缓冲区大小

socket.send.buffer.bytes=102400

#接收套接字的缓冲区大小

socket.receive.buffer.bytes=102400

#请求套接字的缓冲区大小

socket.request.max.bytes=104857600

#kafka运行日志存放的路径

log.dirs=/opt/module/kafka/logs

#topic在当前broker上的分区个数

num.partitions=1

#用来恢复和清理data下数据的线程数量

num.recovery.threads.per.data.dir=1

#segment文件保留的最长时间，超时将被删除

log.retention.hours=168

#配置连接Zookeeper集群地址

zookeeper.connect=hadoop102:2181,hadoop103:2181,hadoop104:2181

**5）**配置环境变量

[user1@hadoop102 module]$ sudo vim /etc/profile.d/my\_env.sh

#KAFKA\_HOME

export KAFKA\_HOME=/opt/module/kafka

export PATH=$PATH:$KAFKA\_HOME/bin

[user1@hadoop102 module]$ source /etc/profile

**6）**分发安装包

[user1@hadoop102 module]$ xsync kafka/

注意：分发之后记得配置其他机器的环境变量

**7）**分别在hadoop103和hadoop104上修改配置文件/opt/module/kafka/config/server.properties中的broker.id=1、broker.id=2

注：broker.id不得重复

**8）**启动集群

先启动Zookeeper集群，然后启动kafaka

[user1@hadoop102 kafka]$ zk.sh start

依次在hadoop102、hadoop103、hadoop104节点上启动kafka

[user1@hadoop102 kafka]$ bin/kafka-server-start.sh -daemon config/server.properties

[user1@hadoop103 kafka]$ bin/kafka-server-start.sh -daemon config/server.properties

[user1@hadoop104 kafka]$ bin/kafka-server-start.sh -daemon config/server.properties

**9）**关闭集群

[user1@hadoop102 kafka]$ bin/kafka-server-stop.sh stop

[user1@hadoop103 kafka]$ bin/kafka-server-stop.sh stop

[user1@hadoop104 kafka]$ bin/kafka-server-stop.sh stop

**10）**kafka群起脚本

#!/bin/bash

if [ $# -lt 1 ]

then

echo "Input Args Error....."

exit

fi

for i in hadoop102 hadoop103 hadoop104

do

case $1 in

start)

echo "==================START $i KAFKA==================="

ssh $i /opt/module/kafka\_2.11-2.4.1/bin/kafka-server-start.sh -daemon /opt/module/kafka\_2.11-2.4.1/config/server.properties

;;

stop)

echo "==================STOP $i KAFKA==================="

ssh $i /opt/module/kafka\_2.11-2.4.1/bin/kafka-server-stop.sh stop

;;

\*)

echo "Input Args Error....."

exit

;;

esac

done

## 2.2 Kafka命令行操作

**1）**查看当前服务器中的所有topic

[user1@hadoop102 kafka]$ bin/kafka-topics.sh --zookeeper hadoop102,hadoop103,hadoop104:2181/kafka --list

**2）**创建topic

[user1@hadoop102 kafka]$ bin/kafka-topics.sh --zookeeper hadoop102,hadoop103,hadoop104:2181/kafka --create --replication-factor 3 --partitions 1 --topic first

选项说明：

--topic 定义topic名

--replication-factor 定义副本数

--partitions 定义分区数

**3）**删除topic

[user1@hadoop102 kafka]$ bin/kafka-topics.sh --zookeeper hadoop102:2181 --delete --topic first

**4）**发送消息

[user1@hadoop102 kafka]$ bin/kafka-console-producer.sh --broker-list hadoop102:9092,hadoop103:9092,hadoop104:9092 --topic first

>hello world

>user1 user1

**5）**消费消息

[user1@hadoop102 kafka]$ bin/kafka-console-consumer.sh \

--bootstrap-server hadoop102:9092 --topic first

[user1@hadoop102 kafka]$ bin/kafka-console-consumer.sh \

--bootstrap-server hadoop102:9092 --from-beginning --topic first

--from-beginning：会把主题中现有的所有的数据都读取出来。

**6）**查看某个Topic的详情

[user1@hadoop102 kafka]$ bin/kafka-topics.sh --zookeeper hadoop102:2181 --describe –

-topic first

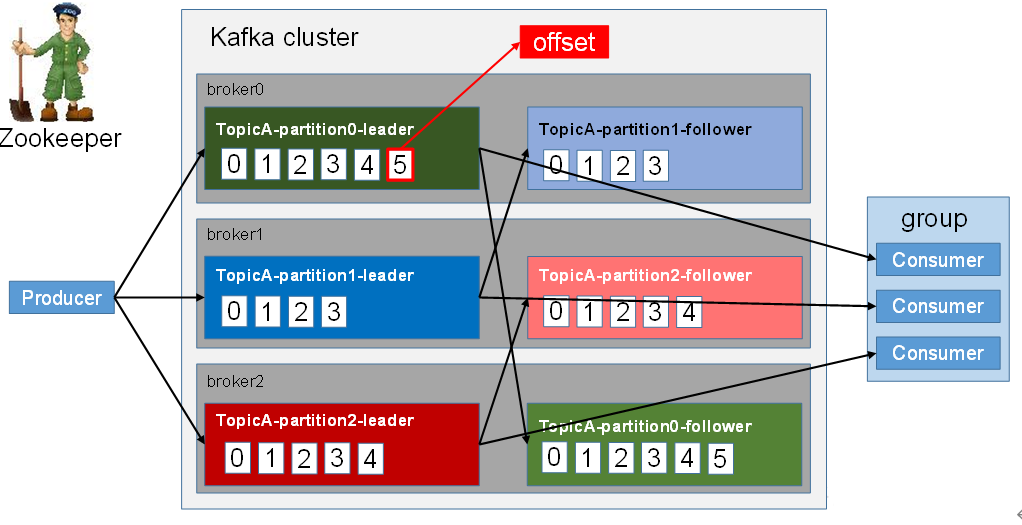
**7）**修改分区数

[user1@hadoop102 kafka]$ bin/kafka-topics.sh --zookeeper hadoop102:2181 --alter –-

topic first --partitions 6

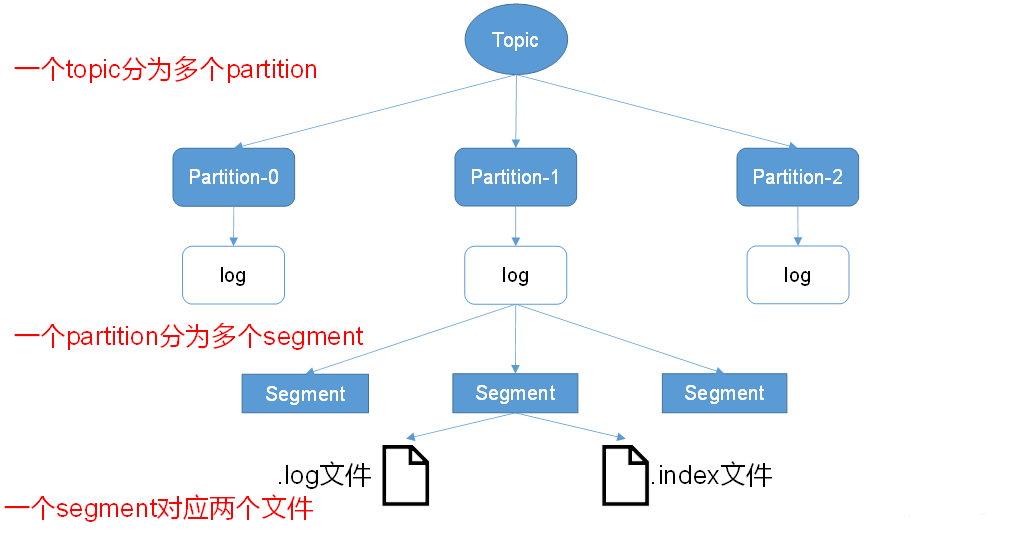
# 第3章 Kafka架构深入

## 3.1 Kafka工作流程及文件存储机制



Kafka中消息是以**topic**进行分类的，生产者生产消息，消费者消费消息，都是面向topic的。

topic是逻辑上的概念，而partition是物理上的概念，每个partition对应于一个log文件，该log文件中存储的就是producer生产的数据。Producer生产的数据会被不断追加到该log文件末端，且每条数据都有自己的offset。消费者组中的每个消费者，都会实时记录自己消费到了哪个offset，以便出错恢复时，从上次的位置继续消费。



由于生产者生产的消息会不断追加到log文件末尾，为防止log文件过大导致数据定位效率低下，Kafka采取了**分片**和**索引**机制，将每个partition分为多个segment。每个segment对应两个文件——“.index”文件和“.log”文件。这些文件位于一个文件夹下，该文件夹的命名规则为：topic名称+分区序号。例如，first这个topic有三个分区，则其对应的文件夹为first-0,first-1,first-2。

00000000000000000000.index

00000000000000000000.log

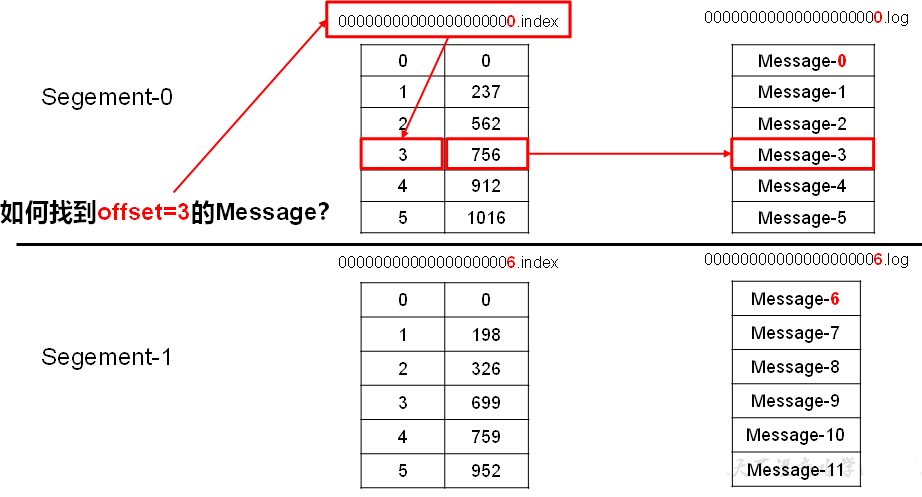
00000000000000170410.index

00000000000000170410.log

00000000000000239430.index

00000000000000239430.log

index和log文件以当前segment的第一条消息的offset命名。下图为index文件和log文件的结构示意图。



“.index”文件存储大量的索引信息，“.log”文件存储大量的数据，索引文件中的元数据指向对应数据文件中message的物理偏移地址。

## 3.2 Kafka生产者

### 3.2.1 分区策略

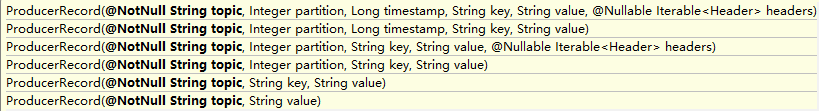
**1）分区的原因**

（1）**方便在集群中扩展**，每个Partition可以通过调整以适应它所在的机器，而一个topic又可以有多个Partition组成，因此整个集群就可以适应任意大小的数据了；

（2）**可以提高并发**，因为可以以Partition为单位读写了。

**2）分区的原则**

我们需要将producer发送的数据封装成一个**ProducerRecord**对象。



（1） 指明 partition 的情况下，直接将指明的值直接作为 partiton 值；

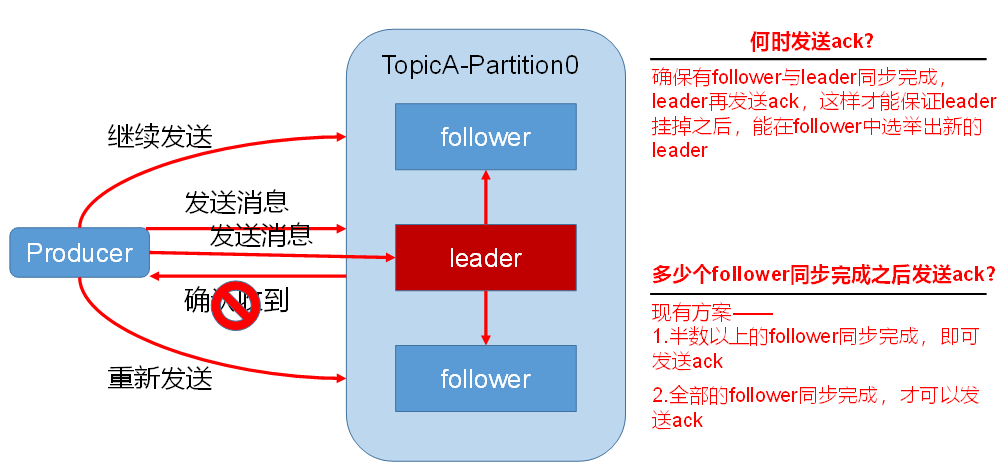
（2） 没有指明 partition 值但有 key 的情况下，将 key 的 hash 值与 topic 的 partition 数进行取余得到 partition 值；

（3） 既没有 partition 值又没有 key 值的情况下， kafka采用Sticky Partition(黏性分区器)，会随机选择一个分区，并尽可能一直使用该分区，待该分区的batch已满或者已完成，kafka再随机一个分区进行使用.

### 3.2.2 数据可靠性保证

**1）生产者发送数据到topic partition的可靠性保证**

为保证producer发送的数据，能可靠的发送到指定的topic，topic的每个partition收到producer发送的数据后，都需要向producer发送ack（acknowledgement确认收到），如果producer收到ack，就会进行下一轮的发送，否则重新发送数据。



**2）Topic partition存储数据的可靠性保证**

（1）**副本数据同步策略**

|  |  |  |
| --- | --- | --- |
| 方案 | 优点 | 缺点 |
| 半数以上完成同步，就发送ack | 延迟低 | 选举新的leader时，容忍n台节点的故障，需要2n+1个副本 |
| 全部完成同步，才发送ack | 选举新的leader时，容忍n台节点的故障，需要n+1个副本 | 延迟高 |

Kafka选择了第二种方案，原因如下：

1. 同样为了容忍n台节点的故障，第一种方案需要2n+1个副本，而第二种方案只需要n+1个副本，而Kafka的每个分区都有大量的数据，第一种方案会造成大量数据的冗余。

2. 虽然第二种方案的网络延迟会比较高，但网络延迟对Kafka的影响较小。

（2）**ISR**

采用第二种方案之后，设想以下情景：leader收到数据，所有follower都开始同步数据，但有一个follower，因为某种故障，迟迟不能与leader进行同步，那leader就要一直等下去，直到它完成同步，才能发送ack。这个问题怎么解决呢？

Leader维护了一个动态的in-sync replica set (ISR)，意为和leader保持同步的follower集合。当ISR中的follower完成数据的同步之后，leader就会给producer发送ack。如果follower长时间未向leader同步数据，则该follower将被踢出ISR，该时间阈值由**replica.lag.time.max.ms**参数设定。Leader发生故障之后，就会从ISR中选举新的leader。

（3）**ack应答级别**

对于某些不太重要的数据，对数据的可靠性要求不是很高，能够容忍数据的少量丢失，所以没必要等ISR中的follower全部接收成功。

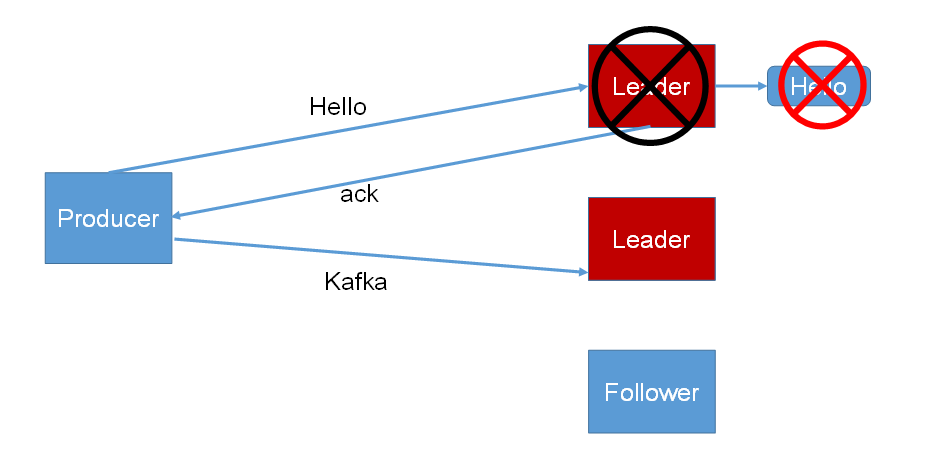
所以Kafka为用户提供了三种可靠性级别，用户根据对可靠性和延迟的要求进行权衡，选择以下的配置。

**acks参数配置：**

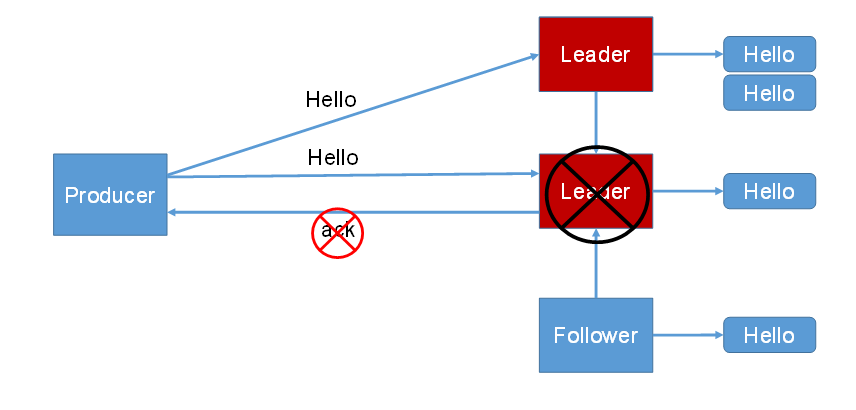
**acks**：

0：这一操作提供了一个最低的延迟，partition的leader接收到消息还没有写入磁盘就已经返回ack，当leader故障时有可能**丢失数据**；

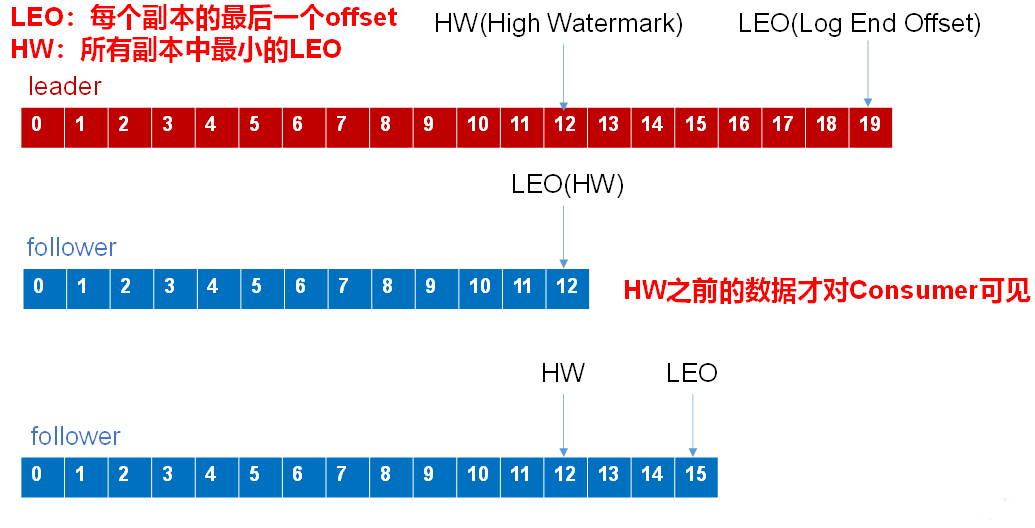
1： partition的leader落盘成功后返回ack，如果在follower同步成功之前leader故障，那么将会**丢失数据**；



-1（all）： partition的leader和follower全部落盘成功后才返回ack。但是如果在follower同步完成后，broker发送ack之前，leader发生故障，那么会造成**数据重复**。



**3）leader和 follower故障处理细节**



**LEO：指的是每个副本最大的offset；**

**HW：指的是消费者能见到的最大的offset，ISR队列中最小的LEO。**

**（1）follower故障**

follower发生故障后会被临时踢出ISR，待该follower恢复后，follower会读取本地磁盘记录的上次的HW，并将log文件高于HW的部分截取掉，从HW开始向leader进行同步。等该**follower的LEO大于等于该Partition的HW**，即follower追上leader之后，就可以重新加入ISR了。

**（2）leader故障**

leader发生故障之后，会从ISR中选出一个新的leader，之后，为保证多个副本之间的数据一致性，其余的follower会先将各自的log文件**高于HW的部分截掉**，然后从新的leader同步数据。

注意：这只能保证副本之间的数据一致性，并不能保证数据不丢失或者不重复。

### 3.2.3 Exactly Once语义

将服务器的ACK级别设置为-1，可以保证Producer到Server之间不会丢失数据，即At Least Once语义。相对的，将服务器ACK级别设置为0，可以保证生产者每条消息只会被发送一次，即At Most Once语义。

**At Least Once + 幂等性 = Exactly Once**

要启用幂等性，只需要将Producer的参数中**enable.idempotence**设置为true即可。

## 3.3 Kafka消费者

### 3.3.1 消费方式

consumer采用pull（拉）模式从broker中读取数据。

push（推）模式很难适应消费速率不同的消费者，因为消息发送速率是由broker决定的。它的目标是尽可能以最快速度传递消息，但是这样很容易造成consumer来不及处理消息，典型的表现就是拒绝服务以及网络拥塞。而pull模式则可以根据consumer的消费能力以适当的速率消费消息。

pull模式不足之处是，如果kafka没有数据，消费者可能会陷入循环中，一直返回空数据。针对这一点，Kafka的消费者在消费数据时会传入一个时长参数timeout，如果当前没有数据可供消费，consumer会等待一段时间之后再返回，这段时长即为timeout。

### 3.3.2 分区分配策略

一个consumer group中有多个consumer，一个 topic有多个partition，所以必然会涉及到partition的分配问题，即确定那个partition由哪个consumer来消费。

Kafka有三种分配策略，RoundRobin，Range , Sticky。

在 Kafka内部存在两种默认的分区分配策略：Range和 RoundRobin。

Range是默认策略。Range是对每个Topic而言的（即一个Topic一个Topic分），首先对同一个Topic里面的分区按照序号进行排序，并对消费者按照字母顺序进行排序。然后用Partitions分区的个数除以消费者线程的总数来决定每个消费者线程消费几个分区。如果除不尽，那么前面几个消费者线程将会多消费一个分区。

例如：我们有10个分区，两个消费者（C1，C2），3个消费者线程，10 / 3 = 3而且除不尽。

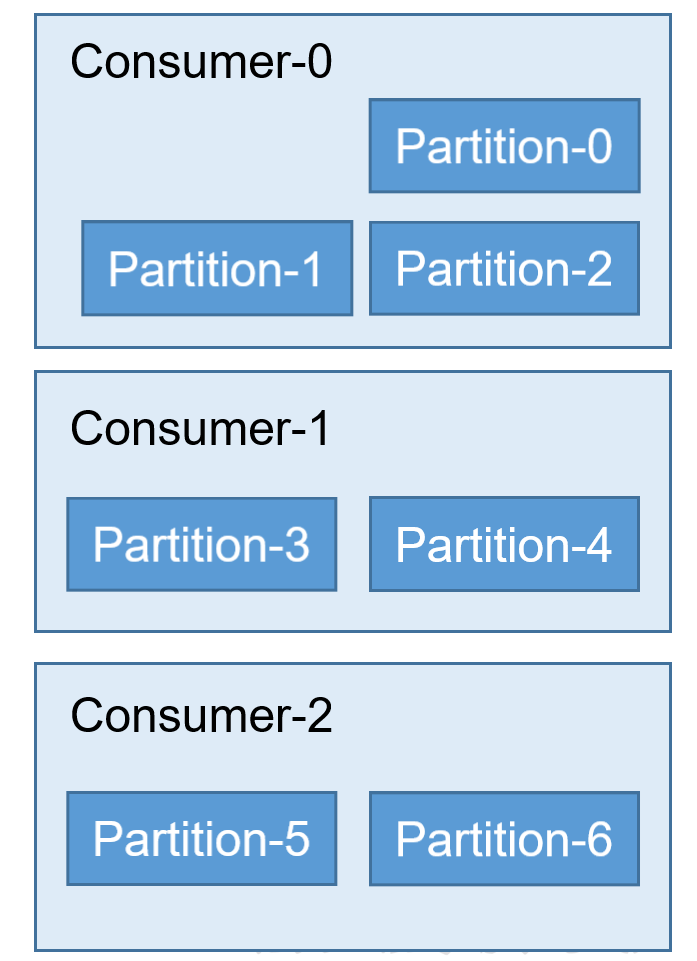
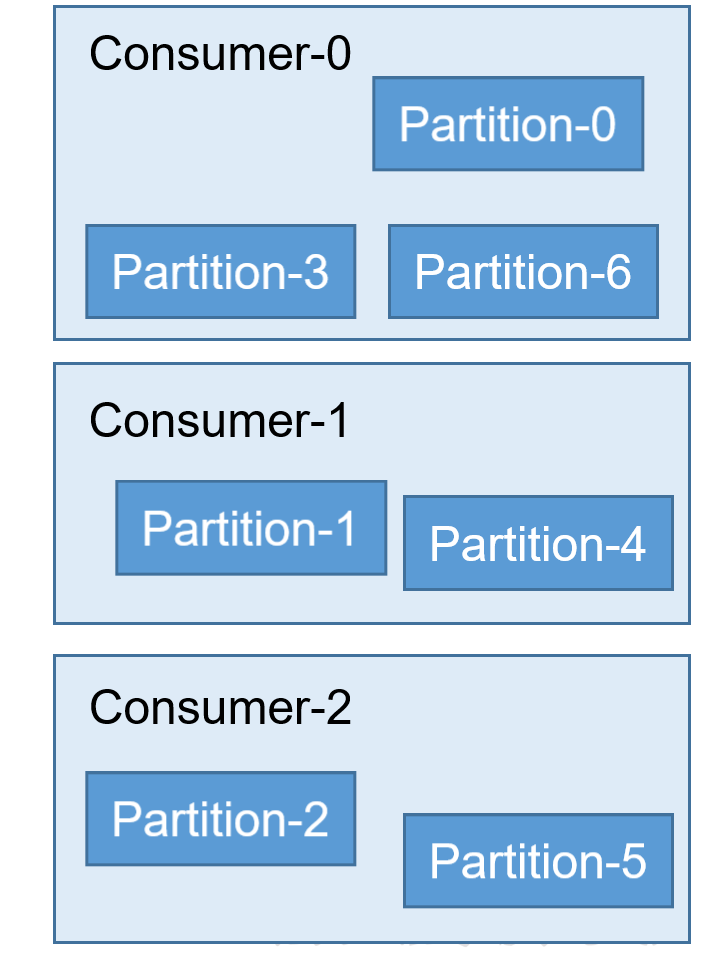
C1-0 将消费 0, 1, 2, 3 分区

C2-0 将消费 4, 5, 6 分区

C2-1 将消费 7, 8, 9 分区

第一步：将所有主题分区组成TopicAndPartition列表，然后对TopicAndPartition列表按照hashCode进行排序，最后按照轮询的方式发给每一个消费线程。

**1）RoundRobin 2）Range**



### 3.3.3 offset的维护

由于consumer在消费过程中可能会出现断电宕机等故障，consumer恢复后，需要从故障前的位置的继续消费，所以consumer需要实时记录自己消费到了哪个offset，以便故障恢复后继续消费。

Kafka 0.9版本之前，consumer默认将offset保存在Zookeeper中，从0.9版本开始，consumer默认将offset保存在Kafka一个内置的topic中，该topic为**\_\_consumer\_offsets**。

**1）消费offset案例**

（0）思想: \_\_consumer\_offsets 为kafka中的topic， 那就可以通过消费者进行消费.

（1）修改配置文件consumer.properties

# 不排除内部的topic

exclude.internal.topics=false

（2）创建一个topic

bin/kafka-topics.sh --create --topic user1 --zookeeper hadoop102:2181 --partitions 2

--replication-factor 2

（3）启动生产者和消费者，分别往user1生产数据和消费数据

bin/kafka-console-producer.sh --topic user1 --broker-list hadoop102:9092

bin/kafka-console-consumer.sh --consumer.config config/consumer.properties --topic user1 --bootstrap-server hadoop102:9092

（4）消费offset

bin/kafka-console-consumer.sh --topic \_\_consumer\_offsets --bootstrap-server hadoop102:9092 --formatter "kafka.coordinator.group.GroupMetadataManager\$OffsetsMessageFormatter" --consumer.config config/consumer.properties --from-beginning

（5）消费到的数据

[test-consumer-group,user1,1]::OffsetAndMetadata(offset=2, leaderEpoch=Optional[0],

metadata=, commitTimestamp=1591935656078, expireTimestamp=None)

[test-consumer-group,user1,0]::OffsetAndMetadata(offset=1, leaderEpoch=Optional[0], metadata=, commitTimestamp=1591935656078, expireTimestamp=None)

## 3.4 Kafka 高效读写数据

**1）顺序写磁盘**

Kafka的producer生产数据，要写入到log文件中，写的过程是一直追加到文件末端，为顺序写。官网有数据表明，同样的磁盘，顺序写能到600M/s，而随机写只有100K/s。这与磁盘的机械机构有关，顺序写之所以快，是因为其省去了大量磁头寻址的时间。

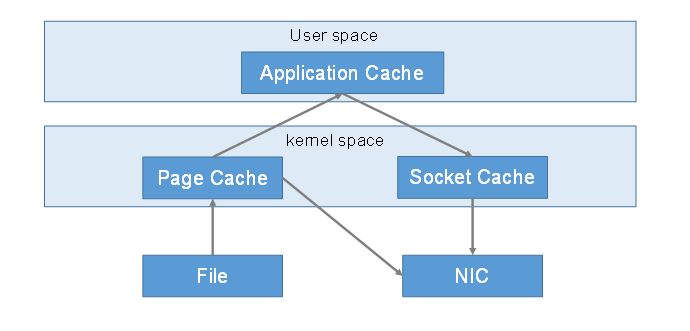
**2）应用**

Kafka数据持久化是直接持久化到Pagecache中，这样会产生以下几个好处：

* I/O Scheduler 会将连续的小块写组装成大块的物理写从而提高性能
* I/O Scheduler 会尝试将一些写操作重新按顺序排好，从而减少磁盘头的移动时间
* 充分利用所有空闲内存（非 JVM 内存）。如果使用应用层 Cache（即 JVM 堆内存），会增加 GC 负担
* 读操作可直接在 Page Cache 内进行。如果消费和生产速度相当，甚至不需要通过物理磁盘（直接通过 Page Cache）交换数据
* 如果进程重启，JVM 内的 Cache 会失效，但 Page Cache 仍然可用

尽管持久化到Pagecache上可能会造成宕机丢失数据的情况，但这可以被Kafka的Replication机制解决。如果为了保证这种情况下数据不丢失而强制将 Page Cache 中的数据 Flush 到磁盘，反而会降低性能。

**3）零复制技术**

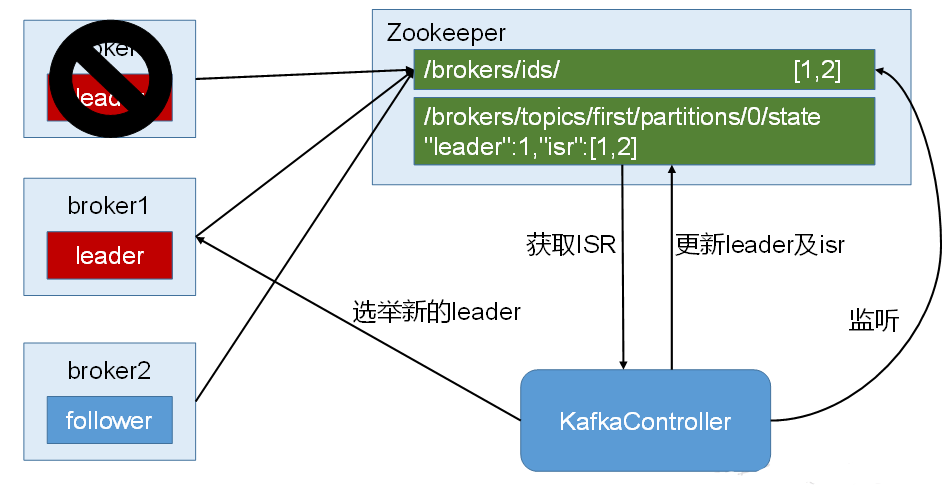


## 3.5 Zookeeper在Kafka中的作用

Kafka集群中有一个broker会被选举为Controller，负责管理集群broker的上下线，所有topic的分区副本分配和leader选举等工作。

Controller的管理工作都是依赖于Zookeeper的。

以下为partition的leader选举过程：



## 3.6 Kafka事务

Kafka从0.11版本开始引入了事务支持。事务可以保证Kafka在Exactly Once语义的基础上，生产和消费可以跨分区和会话，要么`全部成功，要么全部失败。

### 3.6.1 Producer事务

为了实现跨分区跨会话的事务，需要引入一个全局唯一的Transaction ID，并将Producer获得的PID和Transaction ID绑定。这样当Producer重启后就可以通过正在进行的Transaction ID获得原来的PID。

为了管理Transaction，Kafka引入了一个新的组件Transaction Coordinator。Producer就是通过和Transaction Coordinator交互获得Transaction ID对应的任务状态。Transaction Coordinator还负责将事务所有写入Kafka的一个内部Topic，这样即使整个服务重启，由于事务状态得到保存，进行中的事务状态可以得到恢复，从而继续进行。

### 3.6.2 Consumer事务（精准一次性消费）

上述事务机制主要是从Producer方面考虑，对于Consumer而言，事务的保证就会相对较弱，尤其时无法保证Commit的信息被精确消费。这是由于Consumer可以通过offset访问任意信息，而且不同的Segment File生命周期不同，同一事务的消息可能会出现重启后被删除的情况。

如果想完成Consumer端的精准一次性消费，那么需要kafka消费端将消费过程和提交offset过程做原子绑定。此时我们需要将kafka的offset保存到支持事务的自定义介质（比如mysql）。这部分知识会在后续项目部分涉及。

# 第5章 Kafka监控

## 5.1 Kafka Eagle

**1）修改kafka启动命令**

修改kafka-server-start.sh命令中

if [ "x$KAFKA\_HEAP\_OPTS" = "x" ]; then

export KAFKA\_HEAP\_OPTS="-Xmx1G -Xms1G"

fi

为

if [ "x$KAFKA\_HEAP\_OPTS" = "x" ]; then

export KAFKA\_HEAP\_OPTS="-server -Xms2G -Xmx2G -XX:PermSize=128m -XX:+UseG1GC -XX:MaxGCPauseMillis=200 -XX:ParallelGCThreads=8 -XX:ConcGCThreads=5 -XX:InitiatingHeapOccupancyPercent=70"

export JMX\_PORT="9999"

#export KAFKA\_HEAP\_OPTS="-Xmx1G -Xms1G"

fi

注意：修改之后在启动Kafka之前要分发之其他节点

**2）上传压缩包kafka-eagle-bin-1.4.5.tar.gz到集群/opt/software目录**

**3）解压到本地**

[user1@hadoop102 software]$ tar -zxvf kafka-eagle-bin-1.4.5.tar.gz

**4）进入刚才解压的目录**

[user1@hadoop102 kafka-eagle-bin-1.4.5]$ ll

总用量 82932

-rw-rw-r--. 1 user1 user1 84920710 8月 13 23:00 kafka-eagle-web-1.4.5-bin.tar.gz

**5）将kafka-eagle-web-1.3.7-bin.tar.gz解压至/opt/module**

[user1@hadoop102 kafka-eagle-bin-1.4.5]$ tar -zxvf kafka-eagle-web-1.4.5-bin.tar.gz -C /opt/module/

**6）修改名称**

[user1@hadoop102 module]$ mv kafka-eagle-web-1.4.5/ eagle

**7）给启动文件执行权限**

[user1@hadoop102 eagle]$ cd bin/

[user1@hadoop102 bin]$ ll

总用量 12

-rw-r--r--. 1 user1 user1 1848 8月 22 2017 ke.bat

-rw-r--r--. 1 user1 user1 7190 7月 30 20:12 ke.sh

[user1@hadoop102 bin]$ chmod 777 ke.sh

**8）修改配置文件 conf/system-config.properties**

######################################

# multi zookeeper&kafka cluster list

######################################

kafka.eagle.zk.cluster.alias=cluster1

cluster1.zk.list=hadoop102:2181,hadoop103:2181,hadoop104:2181

######################################

# kafka offset storage

######################################

cluster1.kafka.eagle.offset.storage=kafka

######################################

# enable kafka metrics

######################################

kafka.eagle.metrics.charts=true

kafka.eagle.sql.fix.error=false

######################################

# kafka jdbc driver address

######################################

kafka.eagle.driver=com.mysql.jdbc.Driver

kafka.eagle.url=jdbc:mysql://hadoop102:3306/ke?useUnicode=true&characterEncoding=UTF-8&zeroDateTimeBehavior=convertToNull

kafka.eagle.username=root

kafka.eagle.password=123456

**9）添加环境变量**

export KE\_HOME=/opt/module/eagle

export PATH=$PATH:$KE\_HOME/bin

注意：source /etc/profile

**10）启动**

[user1@hadoop102 eagle]$ bin/ke.sh start

... ...

... ...

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Kafka Eagle Service has started success.

\* Welcome, Now you can visit 'http://192.168.202.102:8048/ke'

\* Account:admin ,Password:123456

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* <Usage> ke.sh [start|status|stop|restart|stats] </Usage>

\* <Usage> https://www.kafka-eagle.org/ </Usage>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[user1@hadoop102 eagle]$

注意：启动之前需要先启动ZK以及KAFKA

**11）登录页面查看监控数据**

<http://192.168.202.102:8048/ke>

