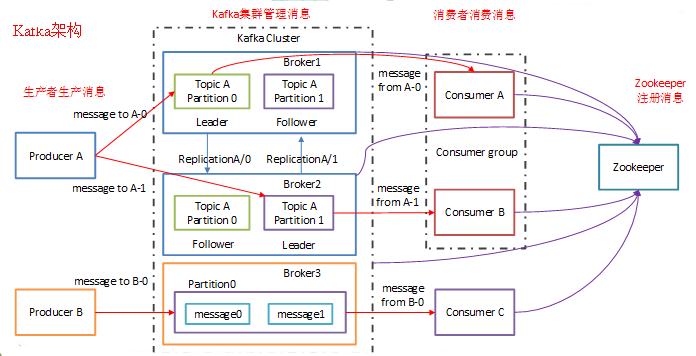
**Kafka的架构**



**Kafka broker**



1. Broker id
2. 主动控制器

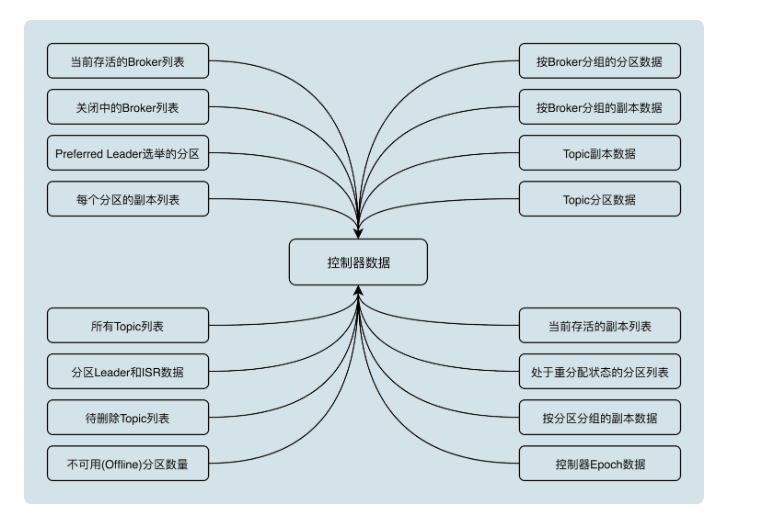
主要作用是在Apache Zookeeper的帮助下管理和协调整个kafka集群（社区正在计划去掉zookeeper的依赖）。集群中的任意一台Broker都可以成为控制器，在一个正常运行的集群中目前只能有一个控制器。

实际上，在Broker启动的时候，会尝试去zookeeper中创建/controller节点。第一个成功创建/controller的节点Broker会被指定为控制器。

控制器的作用：

1. topic管理：主题的创建、删除、增加分区等。当我们在任意一台Broker中执行kafka-topic脚本时，它们会自动找到控制器，并把工作交给控制器来执行。
2. 分区重分配： 分区重分配主要是指，kafka-reassign-partition脚本提供的对已有主题进行细粒度的分配功能。这部分功能也是控制器来实现。
3. Preferred(优先)领导者选举：Preferred领导者选举主要是kafka为了避免部分Broker负载过重而提供的一种换Leader的方案。
4. 集群成员管理：新增Broker、Broker主动关闭、Broker宕机
5. 数据服务： 就是想其他Broker提供数据服务。控制器上保存了最全的集群元数据信息，其他所有Broker会定期接受控制器发来的元数据更新请求，从而更新其内存中的缓存数据。

下面用一张图来说明一下控制器中保存了什么信息：



上面的这些信息，在zookeeper中也保存了一份。每当控制器初始化的时候，它都会从zookeeper上读取对应的元数据并填充到自己的缓存中。有了这些数据，控制器就能对外提供数据库服务了。

**Kafka Topic 和 partition**

1. 创建一个topic

./kafka-topics --create --zookeeper yuntest:2181 --partitions 12 --replica-factors 2 --topic kbtest1

副本数不能大于可用的broker数

1. 删除一个topic

./kafka-topics --delete --zookeeper yuntest:2181 --topic kbtest1

1. 变更一个topic

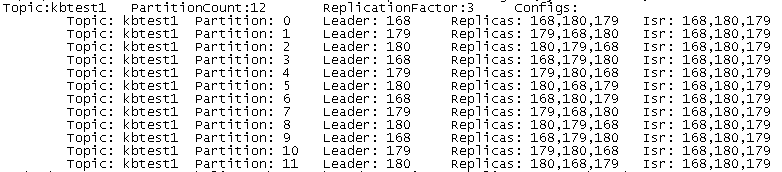
./kafka-topics --alter --zookeeper yuntest:2181 --partitions 15 --replica-factors 3 --topic kbtest1

1. 查看topic 列表

./kafka-topics --list --zookeeper yuntest:2181

1. 查看一个topic的信息

./kafka-topics --describe --zookeeper yuntest:2181 --topic kbtest1



TIPS: ISR(in-sync Replica)，每个Partition都会有一个ISR，而且是由leader动态维护。Replica 被leader踢出isr的条件：

1. 一个节点必须能维持与zookeeper的会话（通过zookeeper的心跳机制）
2. 如果它是一个slave，它必须复制leader并且不能落后"太多"

我们让节点满足这2个“同步”条件，以区分“活着”还是“故障”。leader跟踪“同步”节点。如果一个follower死掉，卡住，或落后，leader将从同步副本列表中移除它。落后是通过replica.lag.max.messages配置控制，卡住是通过replica.lag.time.max.ms配置控制的。

**Kafka分区和副本分配策略**

Kafka 使用 ZK 在 Broker 中选出一个 Controller，用于 Partition 分配和 Leader 选举。Partition 的分配：

    将所有 Broker(假设共 n 个 Broker)和待分配的 Partition 排序。

    将第 i 个 Partition 分配到第(i mod n)个 Broker 上 (这个就是 Leader)。

    将第 i 个 Partition 的第 j 个 Replica 分配到第((i + j) mode n)个 Broker 上。

**Partition leader**

选举流程：

* 1. Leader = -1 ，表示partition处于初始状态或者无leader状态，需要进行leader选举
  2. 初始分配为Replicas列表中的第一个
  3. Leader副本挂了，从isr 中选择第一个，
  4. 若1失败，从replicas选择第一个可以用的,但需要开启一个配置

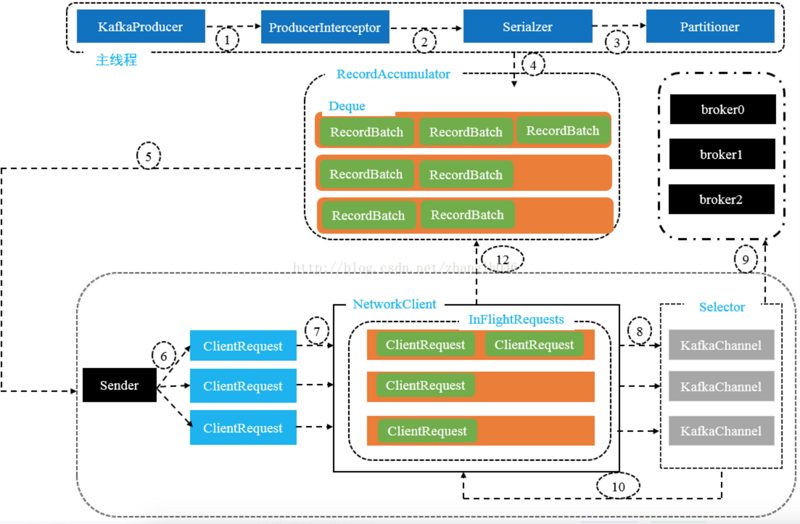


该操作可能会丢失分区数据

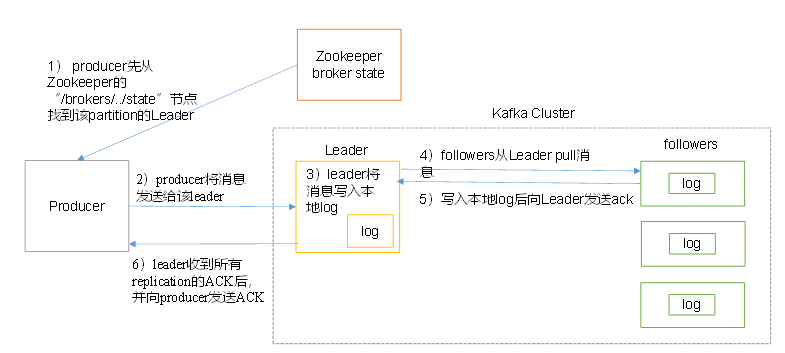
* 1. 若某个partition的副本全部挂掉，就需要做副本重新分配操作

kafka-reassign-partitions.sh

**Producer消息发送流程**



这里步骤9 再细化一下



生产者将数据直接发送到分区leader的broker上（没有任何干预的路由层）。

Question：当生产环境发现生产者发送消息负载很重，直接增加kafka 的broker是否有用？增加topic 的partition数量是否有用？增加生产者线程是否有用？还是有其他的方法？

**Producer 几个常用的参数 （v0.10.0.1）**

*acks*

重要性：高  
类型：String  
默认值："1"

应答数设置。producer只有接收到来自server的acks指定数量的应答，才会认为发送给server的消息记录已送达。该配置项用于控制已发送消息记录的持久性，有以下几种设置值：

acks = 0：表示producer无需等待server端的应答消息，producer将record扔到发送缓冲区，就认为该record已经发送，然后转身走人。这种情况无法保证server端真的成功接收到该消息记录，且此时即使retries配置项也无法生效，因为producer无法知道是否失败。另外，每个record返回的*offset*都被设为-1。

acks = 1：表示接收该消息记录的分区leader将消息记录写入到本地log文件，就返回Acknowledgement，告知producer本次发送已完成，而无需等待其他follower分区的确认。这种情况下，可能出现消息记录没有备份的情况（follower宕机等）。

acks = all：表示消息记录只有得到分区leader以及其他分区副本同步结点队列（ISR）中的分区follower的确认之后，才能回复acknowlegement，告知producer本次发送已完成。这种情况下，只要分区副本同步结点队列（ISR）中某一个follower存活，那么消息记录就不会被丢失。这种方式最安全，但效率也最低。

acks = -1：等同于acks = all。

*bootstrap.servers*

重要性：高  
类型：List  
默认值：Collections.emptyList()

引导producer查找Kafka集群所有broker的*引导服务地址列表*。

顾名思义，该配置项是**引导**服务列表，即用于查找Kafka集群中所有broker的host:port列表，producer通过这些host:port与kafka集群建立连接。producer用该列表中的地址只是用于发现kafka集群中所有的服务*broker*，而在kafka集群中，broker可能是动态改变的。另外，Kafka机制中，可以通过某一个broker而查询到所有其他broker，所以在bootstrap.servers中，并不需要配置所有broker的host:port，理想情况下，只需要配置其中的某一个就可以了。但为了提高可用性，避免因该broker挂掉而导致无法查找，那么可以选择配置多个。

配置格式为：

host1:port1,host2:port2,...

*buffer.memory*

重要性：高  
类型：Long  
默认值：33554432字节，即32M

producer可以用于缓存等待发送到服务端的消息记录的缓冲区大小，当消息记录发送到缓冲区的速度大于传输到server的速度，那么等待发送的消息记录将会放在缓冲区，缓冲区如果满了，那么producer会阻塞max.block.ms指定的毫秒数，超过该毫秒数时，将抛出异常。

注意：该缓冲区的大小设置与整个producer需要使用到的producer大体一致，但是要注意并不是所有的缓冲区都是用来存放待发送的records的，比如还有一部分用于压缩数据（当压缩数据的选项开启），还有一部分用于维护in-flight（正在发送）的请求列表。

*compression.type*

重要性：高  
类型：String  
默认值："none"

producer对数据使用的压缩类型，包括：

* none：无压缩类型
* gzip
* snappy
* lz4

producer在压缩数据时，是对所有batches的数据一起进行压缩，而不是一个batch一个batch压缩，所以，一次压缩的batches越多，压缩率越高，压缩效果越好。

*batch.size*

重要性：高  
类型：Long  
默认值：16384字节，即16K

消息记录batch(批)大小限制。kafka producer在将消息记录record发送到集群时，会尝试将一批要发送到相同partition的消息记录压缩在一起，称之为batch（批）。每次request，其实不是发送一个record，而是发送若干个batch，而每个batch里面可能包含多个record。这样成批成批的发送，减少了网络请求，有助于提升producer客户端和kafka集群服务的性能。

batch.size就是用来设置一个batch的最大字节数byte。当设置为0时，表示完全禁用batch的功能。如果batch.size设置过大，那么可能造成内存浪费，因为每个发送到不同partition的record都需要预先分配一块batch.size大小的内存。

*linger.ms*

重要性：中  
类型：Long  
默认值：0毫秒，表示无延时，立即发送。

延迟发送消息记录的时间，上面及前面文章中也已经提到过，producer在发送消息记录record的时候，会将发送到同一个partition的records压缩在batch中。但通常这只发生在records到达速度快于records发送速度的情况下，很容易理解：如果发送速度大于record到达速度，则每来一个record都会被立即发送出去，根本不存在将多个records压缩为一个的可能。

但很多时候，即便是发送速度大于到达速度，我们也不希望每个record就发送一次，还是希望分批次发送，以减少发送次数，提升producer客户端和服务器端的性能。为此，我们需要人为地加一个*发送延迟限制*，即每次发送之间，存在一定的时间间隔linger.ms，在这段时间内，可能有多个records到达，此时就可以对他们分组压缩，成批次发送。这类似于TCP的拥塞控制方法。

**注意：**

* linger.ms设置了发送延迟的最高时间上限，另一个配置项batch.size也同时控制着发送的时机。如果为某个partition压缩的batch字节数已经达到了batch.size设置的字节数，那么该batch将被立即发送到指定的partition，即使此时延迟时间还没达到linger.ms的设置。
* 同样的，如果延迟的时间已经达到了linger.ms的设置，那么即使压缩累积的batch没有达到batch.size设置的字节数，也会被发送到指定的partition。
* linger.ms是针对每一个发送到partition的request。即不同partition的request并不是同时发送的。
* 延迟以为这性能降低，需要在延迟和性能之间进行平衡，找到一个合适的linger.ms值。

*retries*

重要性：低  
类型：int  
默认值：0，表示不重试

当该值被设置成大于0时，客户端会重新发送消息，并且记录发送失败的错误。注意，该重试配置项和客户端因收到错误而重发是一样的。当retries配置项大于0，且max.in.flight.requests.per.connection配置项的值大于1时，存在将重试记录重新排序的风险，也就是说，消息记录的顺序可能会被打乱。原因是：当两个batch被发送到同一个partition时，如果第一个失败，而第二个成功，那么第一个会被重试，此时第二个batch就排在前面了。

*key.serializer*

重要性：高  
类型：Class  
默认值：无

消息记录key的序列化类。org.apache.kafka.common.serialization.StringSerializer

*value.serializer*

重要性：高  
类型：Class  
默认值：无

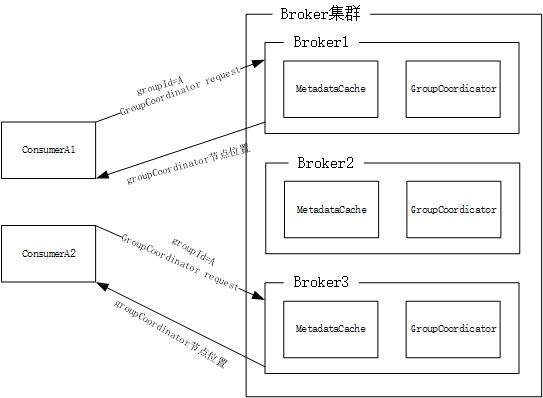
消息记录中value的序列化类。org.apache.kafka.common.serialization.StringSerializer

**Consumer消息消费流程**

Kafka 消费有两种，一种是组消费，一种是直接分区消费。这里只讲组消费流程。

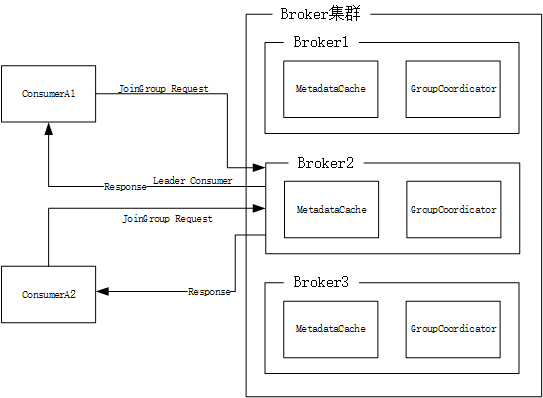
### 第一步：Consumer发送GroupCoordinator请求获取GroupCoordinator所在节点。

如下图，ConsumerA1向Broker1请求GroupCoordinator节点，ConsumerA2向Broker3请求GroupCoordinator节点，Broker集群需要保证Broker1和Broker3分配的是同一个GroupCoordinator（注意一个Broker对应一个GroupCoordinator），这里是通过MedataCache来保证。MedataCache数据在集群中是一致的。



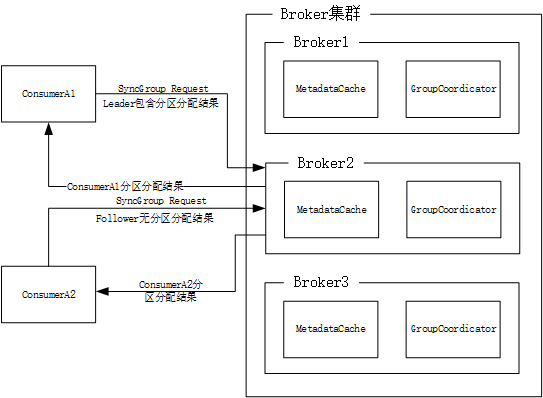
### 第二步：Consumer发送JoinGroup请求加入GroupCoordinator。

假设在第一步中分配结果为Broker2中的GroupCoordinator。此时ConsumerA1和ConsumerA2将分别提交JoinGroup请求加入GroupCoordinator。JoinGroup请求包含消费者的订阅信息、支持的分区分配器、和自定义数据。Broker2中GroupCoordinator收到请求后，会从中选取一个Consumer做为Leader，还会选取一个支持所有Consumer的分区分配策略，然后以JoinGroupResponse返回。图中ConsumerA1被选取作为了Leader。虽然每个消费者都会收到JoinGroupResponse，但是只有 Leader Consumer收到的JoinGroupResponse中封装了其他消费者的信息。当消费者确定自己是Leader后，会根据消费者的信息以及选定的分区分配策略进行分区分配。



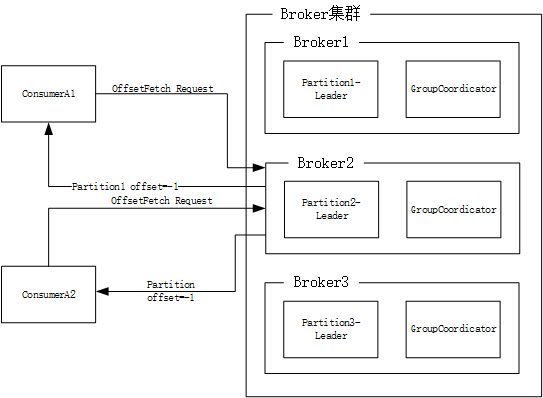
### 第三步：Consumer发送SyncGroup请求获取分区分配。

如图所示，ConsumerA1和ConsumerA2各自发送SyncGroup请求去获取分区分配结果。它们的分配结果是在上一步由ConsumerA1分配的。



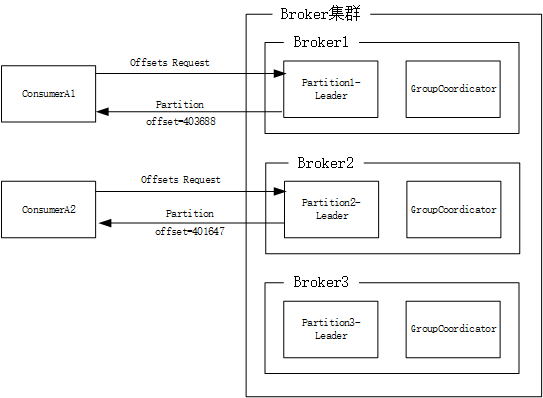
### 第四步：Consumer发送OffsetFetch请求获取分区消费位置。

假设第三步的分配结果为ConsumerA1对应Partition1-Leader，ConsumerA2对应Partition2-Leader。因为ConsumerA1和ConsumerA2从未消费过Partition数据，所以返回的offset为-1。



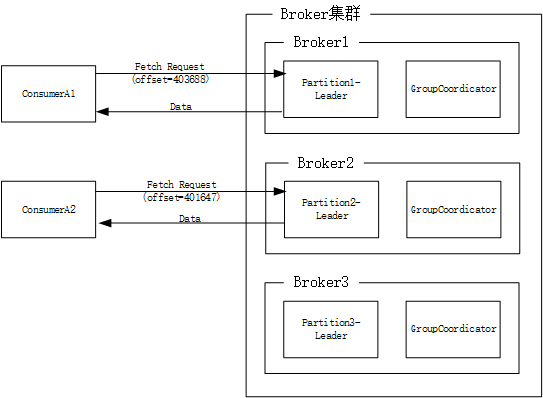
### 第五步：Consumer发送OffsetsRequest请求获取分区消费位置。

因为第四步未得到ConsumerA1和ConsumerA2的消费offset，所以ConsumerA1和ConsumerA2需要发送OffsetsRequest到各自的leader partition所在的broker以至获取起始offset。



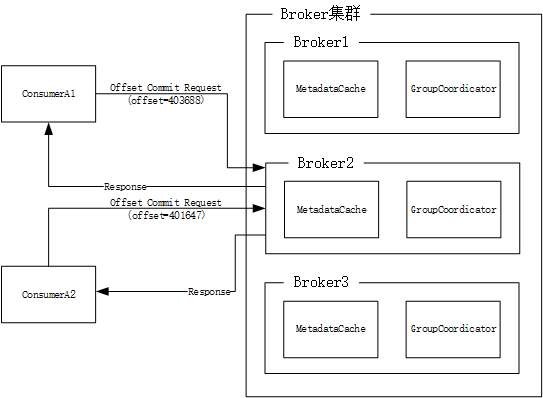
### 第六步：Consumer发送Fetch请求获取数据。

ConsumerA1和ConsumerA2根据上一步获取的offset从各自的Partition Leader中获取数据。



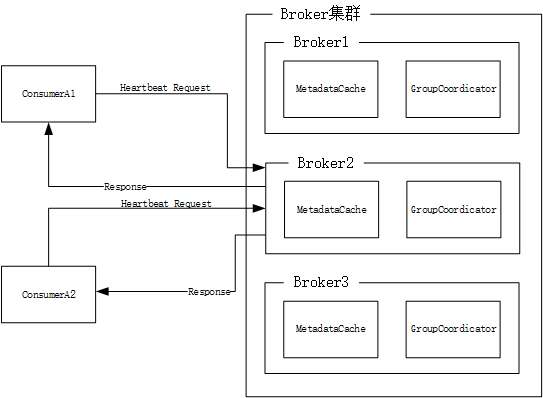
### 第七步：Consumer发送OffsetCommit请求到GroupCoordinator保存消费offset。

保存offset的目的是为了Consumer可以接着上次的消费offset继续消费（比比如Consumer重启）。



### 第八步：Consumer发送Heartbeat请求到GroupCoordinator。

Consumer发送心跳请求到GroupCoordinator。若GroupCoordinator在一个时间段内未检查到Consumer的心跳请求，则GroupCoordinator会要求ConsumerGroup中Consumer的Reblance，进而重新从第二步开始运行。



Offset 存储位置

1. 旧版Kafka 保存在Zookeeper ,查看console-consumer-46965消费组的offset方法

get ${kafka\_root}/consumers/ console-consumer-46965/offsets/topicname/partitionID

1. 新版Kafka已推荐将consumer的位移信息保存在Kafka内部的topic中，即\_\_consumer\_offsets topic，并且默认提供了kafka\_consumer\_groups.sh脚本供用户查看consumer信息。

默认情况下\_\_consumer\_offsets有50个分区，计算指定consumer group在\_\_consumer\_offsets topic中分区方法：

Math.abs(groupID.hashCode()) % numPartitions

如：Math.abs("console-consumer-46965".hashCode()) % 50 = 11，即\_\_consumer\_offsets的分区11保存了这个consumer group的位移信息

获取指定consumer group的位移信息

0.11.0.0版本之前

bin/kafka-simple-consumer-shell.sh --topic \_\_consumer\_offsets **--partition 11** --broker-list localhost:9092,localhost:9093,localhost:9094 --formatter "kafka.coordinator.GroupMetadataManager\$OffsetsMessageFormatter"

0.11.0.0版本以后(含)

bin/kafka-simple-consumer-shell.sh --topic \_\_consumer\_offsets --partition 11 --broker-list localhost:9092,localhost:9093,localhost:9094 --formatter "kafka.coordinator.group.GroupMetadataManager\$OffsetsMessageFormatter"

下面是输出结果：

...

[console-consumer-46965,test,2]::[OffsetMetadata[21,NO\_METADATA],CommitTime 1479092279434,ExpirationTime 1479178679434]

[console-consumer-46965,test,1]::[OffsetMetadata[21,NO\_METADATA],CommitTime 1479092284246,ExpirationTime 1479178684246]

[console-consumer-46965,test,0]::[OffsetMetadata[22,NO\_METADATA],CommitTime 1479092284246,ExpirationTime 1479178684246]

[console-consumer-46965,test,2]::[OffsetMetadata[21,NO\_METADATA],CommitTime 1479092284246,ExpirationTime 1479178684246]

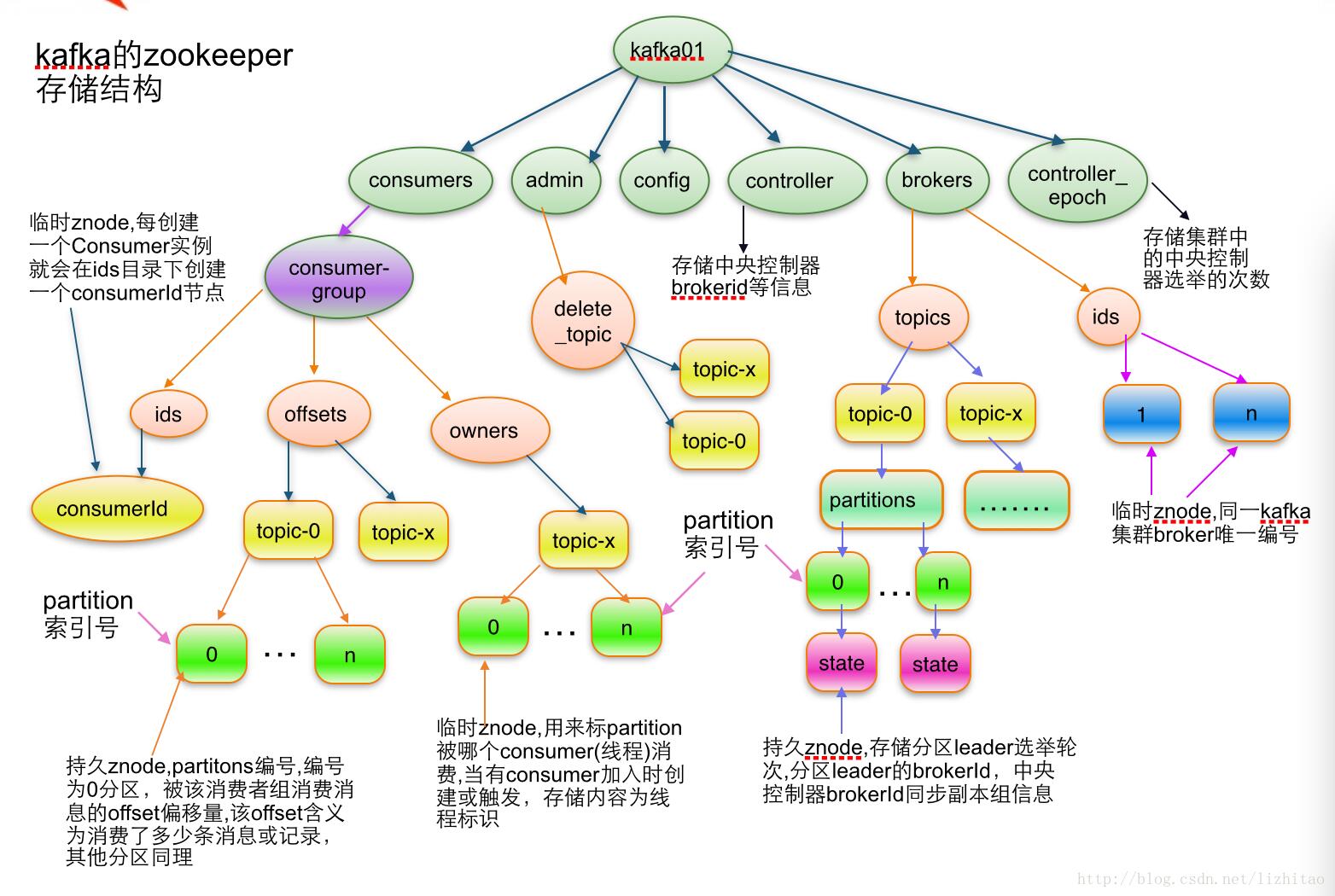
[console-consumer-46965,test,1]::[OffsetMetadata[21,NO\_METADATA],CommitTime 1479092284436,ExpirationTime 1479178684436]

[console-consumer-46965,test,0]::[OffsetMetadata[22,NO\_METADATA],CommitTime 1479092284436,ExpirationTime 1479178684436]

[console-consumer-46965,test,2]::[OffsetMetadata[21,NO\_METADATA],CommitTime 1479092284436,ExpirationTime 1479178684436]

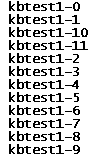
...

**Kafka在zookeeper中数据的存储结构**

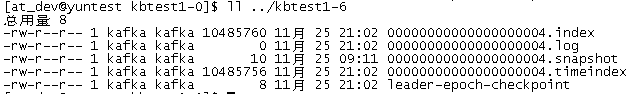


**Kafka log数据文件存储结构**

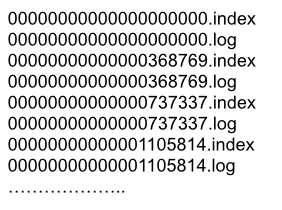
查看存储目录



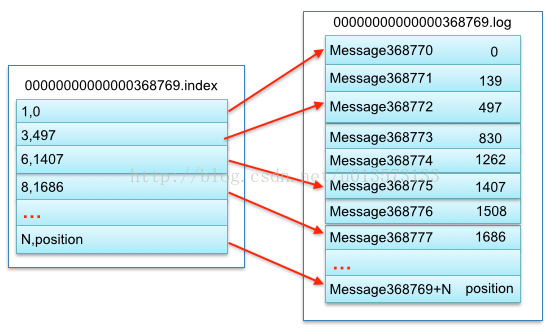
查看目录下的文件



Topic 的log数据文件存储



Index文件和log文件的关系



Kafka的log数据文件存储目录可以设置为多个。

无论消息是否被消费，kafka都会保留所有消息。有两种策略可以删除旧数据：

基于时间：log.retention.ms=168\*3600\*1000

基于大小：log.retention.bytes=1073741824

需要注意的是，因为Kafka读取特定消息的时间复杂度为O(1)，即与文件大小无关，所以这里删除过期文件与提高 Kafka 性能无关。

**维护脚本**

kafka安装目录下的bin目录包含了很多运维可操作的shell脚本，列举如下：

脚本名称 用途描述

connect-distributed.sh 连接kafka集群模式

connect-standalone.sh 连接kafka单机模式

kafka-acls.sh

kafka-broker-api-versions.sh

kafka-configs.sh 配置管理脚本

kafka-console-consumer.sh kafka消费者控制台

kafka-console-producer.sh kafka生产者控制台

kafka-consumer-groups.sh kafka消费者组相关信息

kafka-consumer-perf-test.sh kafka消费者性能测试脚本

kafka-delegation-tokens.sh

kafka-delete-records.sh 删除低水位的日志文件

kafka-log-dirs.sh kafka消息日志目录信息

kafka-mirror-maker.sh 不同数据中心kafka集群复制工具

kafka-preferred-replica-election.sh 触发preferred replica选举

kafka-producer-perf-test.sh kafka生产者性能测试脚本

kafka-reassign-partitions.sh 分区重分配脚本

kafka-replay-log-producer.sh

kafka-replica-verification.sh 复制进度验证脚本

kafka-run-class.sh 所有脚本的基础，都基本调用这个

kafka-server-start.sh 启动kafka服务

kafka-server-stop.sh 停止kafka服务

kafka-simple-consumer-shell.sh deprecated，推荐使用kafka-console-consumer.sh

kafka-streams-application-reset.sh todo

kafka-topics.sh topic管理脚本

kafka-verifiable-consumer.sh 可检验的kafka消费者

kafka-verifiable-producer.sh 可检验的kafka生产者

zookeeper-security-migration.sh todo

zookeeper-server-start.sh 启动zk服务

zookeeper-server-stop.sh 停止zk服务

zookeeper-shell.sh zk客户端

接下来详细说明每个脚本的使用方法。

##### connect-distributed.sh&connect-standalone.sh

Kafka Connect是在0.9以后加入的功能，主要是用来将其他系统的数据导入到Kafka,然后再将Kafka中的数据导出到另外的系统。可以用来做实时数据同步的ETL，数据实时分析处理等。

主要有2种模式：Standalone（单机模式）和Distribute（分布式模式）。

单机主要用来开发，测试，分布式的用于生产环境。

用法比较复杂，建议参考：Kafka Connect教程详解 https://3gods.com/bigdata/Kafka-Connect-Details.html

##### kafka-broker-api-versions.sh

用法：bin/kafka-broker-api-versions.sh –bootstrap-server 10.0.55.229:9092,10.0.55.229:9093,10.0.55.229:9094

##### kafka-configs.sh

配置管理脚本，这个脚本主要分两类用法：describe和alter。

describe相关用法：

查看每个topic的配置：bin/kafka-configs.sh –zookeeper localhost:2181 –describe –entity-type topics

部分结果如下：

Configs for topic 'test123' are

Configs for topic 'TOPIC-TEST-TEST123' are retention.ms=600000

Configs for topic '\_\_consumer\_offsets' are segment.bytes=104857600,cleanup.policy=compact,compression.type=producer

查看broker的配置：bin/kafka-configs.sh –bootstrap-server localhost:9092 –describe –entity-type brokers –entity-name 0

说明：0是broker.id，因为entity-type为brokers，所以entity-name表示broker.id。

alter相关用法：

给指定topic增加配置：bin/kafka-configs.sh –zookeeper localhost:2181 –alter –entity-type topics –entity-name TOPIC-TEST-TEST123 –add-config retention.ms=600000

给指定topic删除配置：bin/kafka-configs.sh –zookeeper localhost:2181 –alter –entity-type topics –entity-name TOPIC-TEST-TEST123 –delete-config max.message.bytes

通过该脚本可以管理的属性，可以通过执行bin/kafka-configs.sh得到的输出中--add-config的desc可以得到。

##### kafka-console-consumer.sh

用法：bin/kafka-console-consumer.sh –bootstrap-server localhost:9092 –topic test123 [–group groupName] [–partition 目标分区]

这个命令后面还可带很多参数：

--key-deserializer：指定key的反序列化方式，默认是 org.apache.kafka.common.serialization.StringDeserializer

--value-deserializer：指定value的反序列化方式，默认是 org.apache.kafka.common.serialization.StringDeserializer

--from-beginning：从最早的消息开始消费，默认是从最新消息开始消费。

--offset： 从指定的消息位置开始消费，如果设置了这个参数，还需要带上--partition。否则会提示：The partition is required when offset is specified.

--timeout-ms：当消费者在这个参数指定时间间隔内没有收到消息就会推出，并抛出异常：kafka.consumer.ConsumerTimeoutException。

--whitelist：接收的topic白名单集合，和--topic二者取其一。例如：--whitelist "test123.\*"（以test123开头的topic），--whitelist "test123"（指定test123这个topic），"test123|fly"（指定test123或者fly这两个topic）。另外一个参数--blacklist用法类似。

##### kafka-console-producer.sh

用法：bin/kafka-console-producer.sh –broker-list localhost:9092 –topic test123

，如果连接集群，那么broker-list参数格式为：HOST1:PORT1,HOST2:PORT2,HOST3:PORT3

这个命令后面还可带很多参数：

--key-serializer：指定key的序列化方式，默认是 org.apache.kafka.common.serialization.StringSerializer

--value-serializer：指定value的序列化方式，默认是 org.apache.kafka.common.serialization.StringSerializer

##### kafka-consumer-groups.sh

查看所有消费者组：bin/kafka-consumer-groups.sh –bootstrap-server localhost:9092 –list

查看某个消费者组：bin/kafka-consumer-groups.sh –bootstrap-server localhost:9092 –group Test123Group –describe，输出结果如下：

Note: This will not show information about old Zookeeper-based consumers.

TOPIC PARTITION CURRENT-OFFSET LOG-END-OFFSET LAG CONSUMER-ID HOST CLIENT-ID

test123 0 8 8 0 consumer-1-7a46c647-8221-4aca-b6bf-ed14571fb0f1 /172.18.36.203 consumer-1

test123 4 10 10 0 consumer-1-7a46c647-8221-4aca-b6bf-ed14571fb0f1 /172.18.36.203 consumer-1

test123 1 8 8 0 consumer-1-7a46c647-8221-4aca-b6bf-ed14571fb0f1 /172.18.36.203 consumer-1

test123 3 6 6 0 consumer-1-7a46c647-8221-4aca-b6bf-ed14571fb0f1 /172.18.36.203 consumer-1

test123 2 9 9 0 consumer-1-7a46c647-8221-4aca-b6bf-ed14571fb0f1 /172.18.36.203 consumer-1

旧版本consumer模式是./kafka-consumer-offset-checker.sh --zookeeper yuntest:2181 --group zf-ud1exec-server --topic zfud1logqueue

##### kafka-consumer-perf-test.sh

perf是performance的缩写，所以这个脚本是kafka消费者性能测试脚本。

用法：bin/kafka-consumer-perf-test.sh –broker-list localhost:9092 –group testGroup –topic test123 –messages 1024

输出结果如下：

start.time, end.time, data.consumed.in.MB, MB.sec, data.consumed.in.nMsg, nMsg.sec, rebalance.time.ms, fetch.time.ms, fetch.MB.sec, fetch.nMsg.sec

2018-07-02 22:49:10:068, 2018-07-02 22:49:12:077, 0.0001, 0.0001, 41, 20.4082, 19, 1990, 0.0001, 20.6030

##### kafka-delete-records.sh

用法：bin/kafka-delete-records.sh –bootstrap-server 10.0.55.229:9092,10.0.55.229:9093,10.0.55.229:9094 –offset-json-file offset.json，offset.json文件内容：

{

"partitions": [{

"topic": "test123",

"partition": 3,

"offset": 10

}],

"version": 1

}

执行结果如下，表示删除test123这个topic下分区为3的offset少于10的消息日志（不会删除offset=10的消息日志）：

Executing records delete operation

Records delete operation completed:

partition: test123-3 low\_watermark: 10

##### kafka-log-dirs.sh

用法：bin/kafka-log-dirs.sh –bootstrap-server localhost:9092 –describe –topic-list test123[,topicName2,topicNameN]，如果没有指定--topic-list，那么会输出所有kafka消息日志目录以及目录下所有topic信息。加上--topic-list参数后，输出结果如下，由这段结果可知，消息日志所在目录为/data/kafka-logs，并且test123这个topic有3个分区：

{

"version": 1,

"brokers": [{

"broker": 0,

"logDirs": [{

"logDir": "/data/kafka-logs",

"error": null,

"partitions": [{

"partition": "test123-1",

"size": 567,

"offsetLag": 0,

"isFuture": false

}, {

"partition": "test123-2",

"size": 639,

"offsetLag": 0,

"isFuture": false

}, {

"partition": "test123-0",

"size": 561,

"offsetLag": 0,

"isFuture": false

}]

}]

}]

}

##### kafka-preferred-replica-election.sh

用法：bin/kafka-preferred-replica-election.sh –zookeeper 10.0.55.208:2181/wallet,10.0.55.209:2181/wallet,10.0.55.210:2181/wallet –path-to-json-file test123-preferred.json（如果不带–path-to-json-file就是对所有topic进行preferred replica election），json文件内容如下：：

{

"partitions": [{

"topic": "test123",

"partition": 0

},

{

"topic": "test123",

"partition": 1

},

{

"topic": "test123",

"partition": 2

}]

}

场景：在创建一个topic时，kafka尽量将partition均分在所有的brokers上，并且将replicas也均分在不同的broker上。每个partitiion的所有replicas叫做”assigned replicas”，”assigned replicas”中的第一个replicas叫”preferred replica”，刚创建的topic一般”preferred replica”是leader。leader replica负责所有的读写。其他replica只是冷备状态，不接受读写请求。但随着时间推移，broker可能会主动停机甚至客观宕机，会发生leader选举迁移，导致机群的负载不均衡。我们期望对topic的leader进行重新负载均衡，让partition选择”preferred replica”做为leader。

kafka提供了一个参数auto.leader.rebalance.enable自动做这件事情，且默认为true，原理是一个后台线程检查并触发leader balance。但是并不建议把这个参数设置为true。因为担心这个自动选举发生在业务高峰期，从而导致影响业务。

验证：

操作比较简单，常见一个3个分区3个副本的topic，然后kill掉一个broker。这时候topic信息如下，我们可以看到broker.id为0的broker上有两个leader：

Topic:test123 PartitionCount:3 ReplicationFactor:3 Configs:

Topic: test123 Partition: 0 Leader: 0 Replicas: 0,1,2 Isr: 0,1,2

Topic: test123 Partition: 1 Leader: 1 Replicas: 1,2,0 Isr: 0,1,2

Topic: test123 Partition: 2 Leader: 0 Replicas: 2,0,1 Isr: 0,1,2

执行kafka-preferred-replica-election.sh脚本后，topic信息如下，leader均匀分布在3个不同的broker上，

Topic:test123 PartitionCount:3 ReplicationFactor:3 Configs:

Topic: test123 Partition: 0 Leader: 0 Replicas: 0,1,2 Isr: 0,1,2

Topic: test123 Partition: 1 Leader: 1 Replicas: 1,2,0 Isr: 0,1,2

Topic: test123 Partition: 2 Leader: 2 Replicas: 2,0,1 Isr: 0,1,2

##### kafka-producer-perf-test.sh

perf是performance的缩写，所以这个脚本是kafka生产者性能测试脚本。

##### kafka-reassign-partitions.sh

场景：将一些topic上的分区从当前所在broker移到其他比如新增的broker上。假设有个名为ORDER-DETAIL的topic，在broker.id为2的broker上：

Topic:ORDER-DETAIL PartitionCount:1 ReplicationFactor:1 Configs:

Topic: ORDER-DETAIL Partition: 0 Leader: 2 Replicas: 2 Isr: 2

现在想要把它移动到broker.id为1的broker上，执行脚本：bin/kafka-reassign-partitions.sh –zookeeper 10.0.55.208:2181/wallet,10.0.55.209:2181/wallet,10.0.55.210:2181/wallet –topics-to-move-json-file move.json –broker-list “1” –generate

--generate参数表示生成一个分区再分配配置，并不会真正的执行，命令执行结果如下：

Current partition replica assignment

{"version":1,"partitions":[{"topic":"ORDER-DETAIL","partition":0,"replicas":[2],"log\_dirs":["any"]}]}

Proposed partition reassignment configuration

{"version":1,"partitions":[{"topic":"ORDER-DETAIL","partition":0,"replicas":[1],"log\_dirs":["any"]}]}

我们只需要把第二段json内容保存到一个新建的final.json文件中（如果知道如何编写这段json内容，那么也可以不执行第一条命令），然后执行命令：bin/kafka-reassign-partitions.sh –zookeeper 10.0.55.208:2181/wallet,10.0.55.209:2181/wallet,10.0.55.210:2181/wallet –reassignment-json-file move\_final.json –execute，此次执行的命令带有--execute参数，说明是真正的执行分区重分配。

通过这个命令还可以给某个topic\*\*增加副本\*\*，例如有一个名为ORDER-DETAIL的topic，有3个分区，但是只有1个副本，为了高可用，需要将副本数增加到2，那么编写replica.json文本内容如下：

{

"version": 1,

"partitions": [{

"topic": "ORDER-DETAIL",

"partition": 0,

"replicas": [1, 2]

},

{

"topic": "ORDER-DETAIL",

"partition": 1,

"replicas": [0, 2]

},

{

"topic": "ORDER-DETAIL",

"partition": 2,

"replicas": [1, 0]

}]

}

然后执行命令即可：bin/kafka-reassign-partitions.sh –zookeeper 10.0.55.208:2181/wallet,10.0.55.209:2181/wallet,10.0.55.210:2181/wallet –reassignment-json-file replica.json

##### kafka-replica-verification.sh

用法：bin/kafka-replica-verification.sh –broker-list 10.0.55.229:9092,10.0.55.229:9093,10.0.55.229:9094 [–topic-white-list test123]，参数–topic-white-list指定要检查的目标topic。输出结果如下，如果输出信息为max lag is 0 for ...表示这个topic的复制没有任何延迟：

2018-07-03 15:04:46,889: verification process is started.

2018-07-03 15:05:16,811: max lag is 0 for partition multi-1 at offset 0 among 5 partitions

2018-07-03 15:05:46,812: max lag is 0 for partition multi-1 at offset 0 among 5 partitions

... ...

##### kafka-server-start.sh

用法：bin/kafka-server-start.sh -daemon config/server.properties，指定配置文件并以守护进程模式启动。

##### kafka-server-stop.sh

用法：bin/kafka-server-stop.sh 。说明，这个命令会kill掉当前服务器上所有kafka broker。但是这个脚本可能执行结果为：No kafka server to stop

##### kafka-simple-consumer-shell.sh

deprecated，用法：bin/kafka-simple-consumer-shell.sh –broker-list 10.0.55.229:9092,10.0.55.229:9093,10.0.55.229:9094 –topic test123

##### kafka-topics.sh

创建topic: bin/kafka-topics.sh –zookeeper localhost:2181 –create –topic test123 –partitions 3 –replication-factor 1

删除topic: bin/kafka-topics.sh –zookeeper localhost:2181 –delete –topic test

修改topic: bin/kafka-topics.sh –zookeeper localhost:2181 –alter –topic test123 –partitions 5，修改topic时只能增加分区数量。

查询topic: bin/kafka-topics.sh –zookeeper localhost:2181 –describe [ –topic test123 ]，查询时如果带上--topic topicName，那么表示只查询该topic的详细信息。这时候还可以带上--unavailable-partitions 和--under-replicated-partitions任意一个参数。

##### kafka-verifiable-consumer.sh

用法：bin/kafka-verifiable-consumer.sh –broker-list 10.0.55.229:9092,10.0.55.229:9093,10.0.55.229:9094 –topic test123 –group-id groupName

这个脚本的作用是接收指定topic的消息消费，并发出消费者事件，例如：offset提交等。

##### kafka-verifiable-producer.sh

用法：bin/kafka-verifiable-producer.sh –broker-list 10.0.55.229:9092,10.0.55.229:9093,10.0.55.229:9094 –topic test123 [–max-messages 64]，建议使用该脚本时增加参数--max-messages，否则会不停的发送消息。

这个脚本的作用是持续发送消息到指定的topic中，参数--max-messages限制最大发送消息数。且每条发送的消息都会有响应信息，这就是和kafka-console-producer.sh最大的不同：

[mwopr@jtcrtvdra35 kafka\_2.12-1.1.0]$ bin/kafka-verifiable-producer.sh --broker-list 10.0.55.229:9092,10.0.55.229:9093,10.0.55.229:9094 --topic test123 --max-messages 9

{"timestamp":1530515959900,"name":"startup\_complete"}

{"timestamp":1530515960310,"name":"producer\_send\_success","key":null,"value":"1","offset":5,"topic":"test123","partition":0}

{"timestamp":1530515960315,"name":"producer\_send\_success","key":null,"value":"4","offset":6,"topic":"test123","partition":0}

{"timestamp":1530515960315,"name":"producer\_send\_success","key":null,"value":"7","offset":7,"topic":"test123","partition":0}

{"timestamp":1530515960316,"name":"producer\_send\_success","key":null,"value":"0","offset":5,"topic":"test123","partition":1}

{"timestamp":1530515960316,"name":"producer\_send\_success","key":null,"value":"3","offset":6,"topic":"test123","partition":1}

{"timestamp":1530515960316,"name":"producer\_send\_success","key":null,"value":"6","offset":7,"topic":"test123","partition":1}

{"timestamp":1530515960316,"name":"producer\_send\_success","key":null,"value":"2","offset":6,"topic":"test123","partition":2}

{"timestamp":1530515960316,"name":"producer\_send\_success","key":null,"value":"5","offset":7,"topic":"test123","partition":2}

{"timestamp":1530515960316,"name":"producer\_send\_success","key":null,"value":"8","offset":8,"topic":"test123","partition":2}

{"timestamp":1530515960333,"name":"shutdown\_complete"}

{"timestamp":1530515960334,"name":"tool\_data","sent":9,"acked":9,"target\_throughput":-1,"avg\_throughput":20.689655172413794}

test123这个topic有3个分区，使用kafka-verifiable-producer.sh发送9条消息。根据输出结果可以看出，往每个分区发送了3条消息。另外，我们可以通过设置参数--max-messages一个比较大的值，可以压测一下搭建的kafka集群环境。

##### zookeeper-shell.sh

用法：bin/zookeeper-shell.sh localhost:2181[/path]，如果kafka集群的zk配置了chroot路径，那么需要加上/path，例如bin/zookeeper-shell.sh localhost:2181/mykafka，登陆zk后，就可以查看kafka写在zk上的节点信息。例如查看有哪些broker，以及broker的详细信息：

ls /brokers/ids

[0]

get /brokers/ids/0

{"listener\_security\_protocol\_map":{"PLAINTEXT":"PLAINTEXT"},"endpoints":["PLAINTEXT://izwz91rhzhed2c54e6yx87z:9092"],"jmx\_port":-1,"host":"izwz91rhzhed2c54e6yx87z","timestamp":"1530435834272","port":9092,"version":4}

cZxid = 0x2d3

ctime = Sun Jul 01 17:03:54 CST 2018

mZxid = 0x2d3

mtime = Sun Jul 01 17:03:54 CST 2018

pZxid = 0x2d3

cversion = 0

dataVersion = 0

aclVersion = 0

ephemeralOwner = 0x1642cb09421006c

dataLength = 216

numChildren = 0