1.kafka启动: ./kafka-server-start.sh ../config/server.properties &

2.创建topic: ./kafka-topics.sh --create --bootstrap-server localhost:2181 --replication-factor 1 --partitions 1 --topic test

3.查看kafka的topic：./kafka-topics.sh --zookeeper master:2181 --list

4.查看kafka某个topic下partition信息: ./kafka-topics.sh --describe --zookeeper master:2181 --topic test-topic

5.查看kafka的指定topic: ./kafka-topics.sh --zookeeper master:2181 --describe --topic yq20171220

6.控制台向kafka生产数据: ./kafka-console-producer.sh --broker-list localhost:9092 --topic test

7.控制台消费kafka的数据: ./kafka-console-consumer.sh --zookeeper socalhost:9092 --topic topic-test --from-beginning

8.查看topic下某分区偏移量的最小值: ./kafka-run-class.sh kafka.tools.GetOffsetShell --topic test- --time -1 --broker-list localhost:9092 --partitions 0

9.增加topic的partition:/kafka-topics.sh --alter --topic jason\_20180519 --zookeeper 10.200.10.24:2181,10.200.10.26:2181,10.200.10.29:2181 --partitions 5

10.删除topic，慎用，只会删除zookeeper中的元数据，消息文件须手动删除: ./kafka-run-class.sh kafka.admin.DeleteTopicCommand --zookeeper master:2181 --topic yq20171220

11.彻底删除topic: rmr /brokers/topics/【topic name】即可

12. 为topic增加partition bin/kafka-topics.sh –zookeeper127.0.0.1:2181–alter –partitions20–topic testKJ1

## 查询集群描述

bin/kafka-topics.sh --describe --zookeeper

## 消费者列表查询

bin/kafka-topics.sh --zookeeper 127.0.0.1:2181 --list

## 新消费者列表查询（支持0.9版本+）

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

## 显示某个消费组的消费详情（仅支持offset存储在zookeeper上的）

bin/kafka-run-class.sh kafka.tools.ConsumerOffsetChecker --zookeeper localhost:2181 --group test

## 显示某个消费组的消费详情（0.9版本 - 0.10.1.0 之前）

bin/kafka-consumer-groups.sh --new-consumer --bootstrap-server localhost:9092 --describe --group test-consumer-group

## 显示某个消费组的消费详情（0.10.1.0版本+）

bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group my-group

##平衡leader

bin/kafka-preferred-replica-election.sh --zookeeper zk\_host:port/chroot

#kafka自带压测命令

bin/kafka-producer-perf-test.sh --topic test --num-records 100 --record-size 1 --throughput 100 --producer-props bootstrap.servers=localhost:9092

## 生产者

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

## 消费者

bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic test

## 新生产者（支持0.9版本+）

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

## 新消费者（支持0.9版本+）

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

## 高级点的用法

bin/kafka-simple-consumer-shell.sh --brist localhost:9092 --topic test --partition 0 --offset 1234 --max-messages 10

下载地址：

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

解压

tar -zxvf kafka\_2.11-0.8.2.0.tgz -C apps

修改 config/server.properties

//当前机器在集群中的唯一标识，和zookeeper的myid性质一样

broker.id=0

//当前kafka对外提供服务的端口默认是9092

port=9092

//存储路径

log.dirs=tmp/kafka-logs

//设置zookeeper的连接端口

zookeeper.connect=192.168.123.102:2181,192.168.123.103:2181,192.168.123.104:2181

启动

./kafka-server-start.sh ../config/server.properties

伪分布式安装

server.properties另存为server0.properties、server1.properties、server2.properties

分别修改

-----------------------------------

broker.id=0

log.dirs=tmp/kafka-logs-0

-----------------------------------

broker.id=1

log.dirs=tmp/kafka-logs-1

-----------------------------------

broker.id=2

log.dirs=tmp/kafka-logs-2

# **重平衡**

分别启动/kafka-server-start.sh ../config/server0.properties

消费组与分区重平衡

可以看到，当新的消费者加入消费组，它会消费一个或多个分区，而这些分区之前是由其他消费者负责的；另外，当消费者离开消费组（比如重启、宕机等）时，它所消费的分区会分配给其他分区。这种现象称为重平衡（rebalance）。重平衡是Kafka一个很重要的性质，这个性质保证了高可用和水平扩展。不过也需要注意到，在重平衡期间，所有消费者都不能消费消息，因此会造成整个消费组短暂的不可用。而且，将分区进行重平衡也会导致原来的消费者状态过期，从而导致消费者需要重新更新状态，这段期间也会降低消费性能。后面我们会讨论如何安全的进行重平衡以及如何尽可能避免。

消费者通过定期发送心跳（hearbeat）到一个作为组协调者（group coordinator）的broker来保持在消费组内存活。这个broker不是固定的，每个消费组都可能不同。当消费者拉取消息或者提交时，便会发送心跳。

如果消费者超过一定时间没有发送心跳，那么它的会话（session）就会过期，组协调者会认为该消费者已经宕机，然后触发重平衡。可以看到，从消费者宕机到会话过期是有一定时间的，这段时间内该消费者的分区都不能进行消息消费；通常情况下，我们可以进行优雅关闭，这样消费者会发送离开的消息到组协调者，这样组协调者可以立即进行重平衡而不需要等待会话过期。

在0.10.1版本，Kafka对心跳机制进行了修改，将发送心跳与拉取消息进行分离，这样使得发送心跳的频率不受拉取的频率影响。另外更高版本的Kafka支持配置一个消费者多长时间不拉取消息但仍然保持存活，这个配置可以避免活锁（livelock）。活锁，是指应用没有故障但是由于某些原因不能进一步消费。

# **Java API:**

<https://memorynotfound.com/spring-kafka-consume-producer-example/>

<https://www.cnblogs.com/wangzhuxing/p/10099894.html#_label8>

<https://blog.csdn.net/wangzhanzheng/article/details/80801059>

<http://kafka.apache.org/10/javadoc/?org/apache/kafka/clients/consumer/KafkaConsumer.html>

消息存储原理

<https://www.cnblogs.com/wxd0108/p/6518762.html>

http://kafka.apache.org/0100/documentation.html#log

每一条消息格式

On-disk format of a message

offset : 8 bytes

message length : 4 bytes (value: 4 + 1 + 1 + 8(if magic value > 0) + 4 + K + 4 + V)

crc : 4 bytes

magic value : 1 byte

attributes : 1 byte

timestamp : 8 bytes (Only exists when magic value is greater than zero)

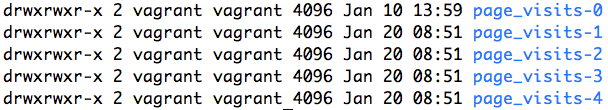
key length : 4 bytes

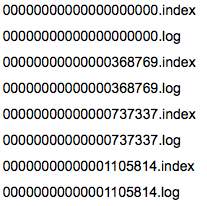
key : K bytes

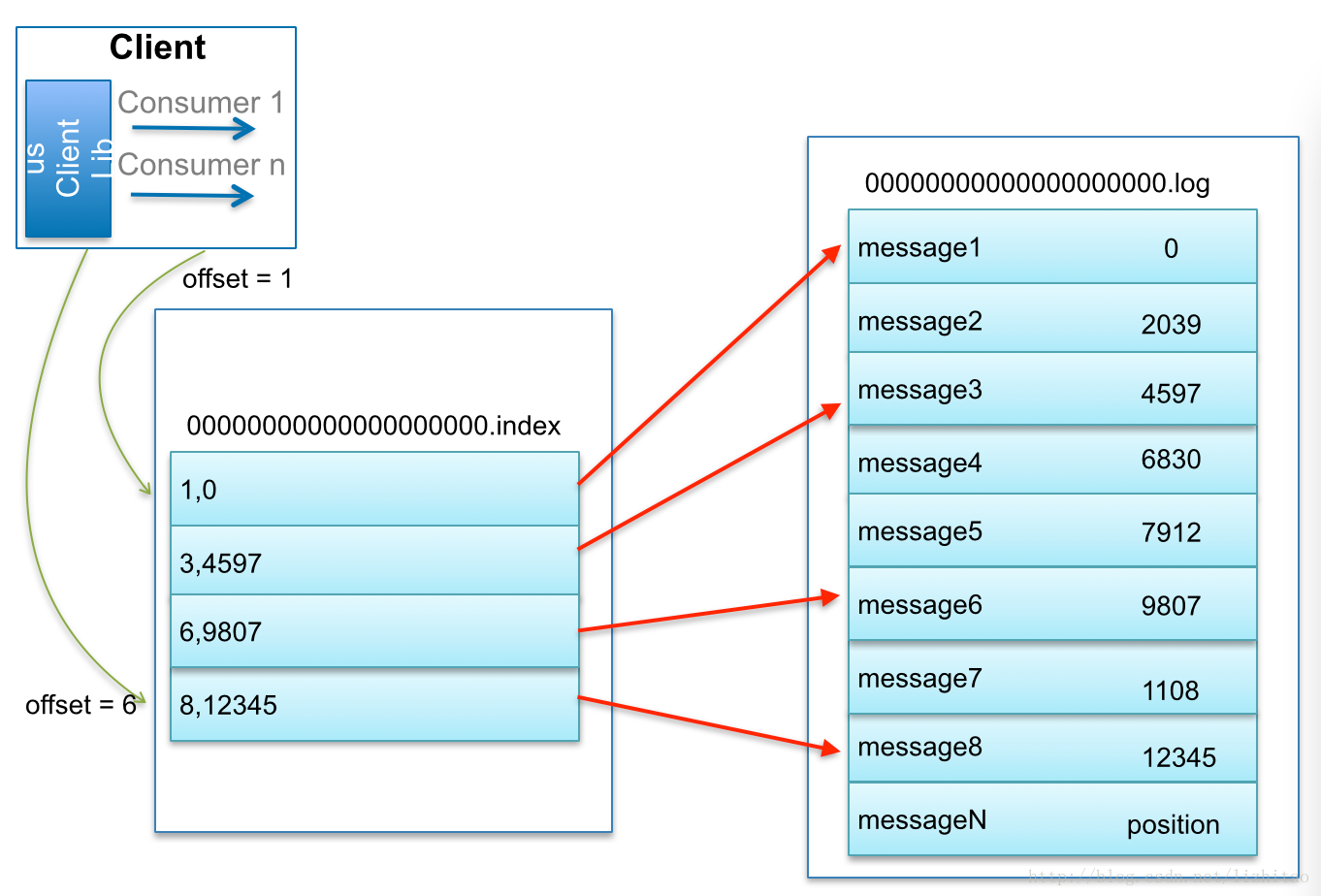
value length : 4 bytes

value : V bytes

我们以几张图来总结一下Message是如何在Kafka中存储的，以及如何查找指定offset的Message的。

Message是按照topic来组织，每个topic可以分成多个的partition，比如：有5个partition的名为为page\_visits的topic的目录结构为：   


partition是分段的，每个段叫LogSegment，包括了一个数据文件和一个索引文件，下图是某个partition目录下的文件：   
   
可以看到，这个partition有4个LogSegment。

借用博主[@lizhitao](http://blog.csdn.net/lizhitao/)博客上的一张图来展示是如何查找Message的。   
  
比如：要查找绝对offset为7的Message：   
1. 首先是用二分查找确定它是在哪个LogSegment中，自然是在第一个Segment中。   
2. 打开这个Segment的index文件，也是用二分查找找到offset小于或者等于指定offset的索引条目中最大的那个offset。自然offset为6的那个索引是我们要找的，通过索引文件我们知道offset为6的Message在数据文件中的位置为9807。   
3. 打开数据文件，从位置为9807的那个地方开始顺序扫描直到找到offset为7的那条Message。

这套机制是建立在offset是有序的。索引文件被映射到内存中，所以查找的速度还是很快的。