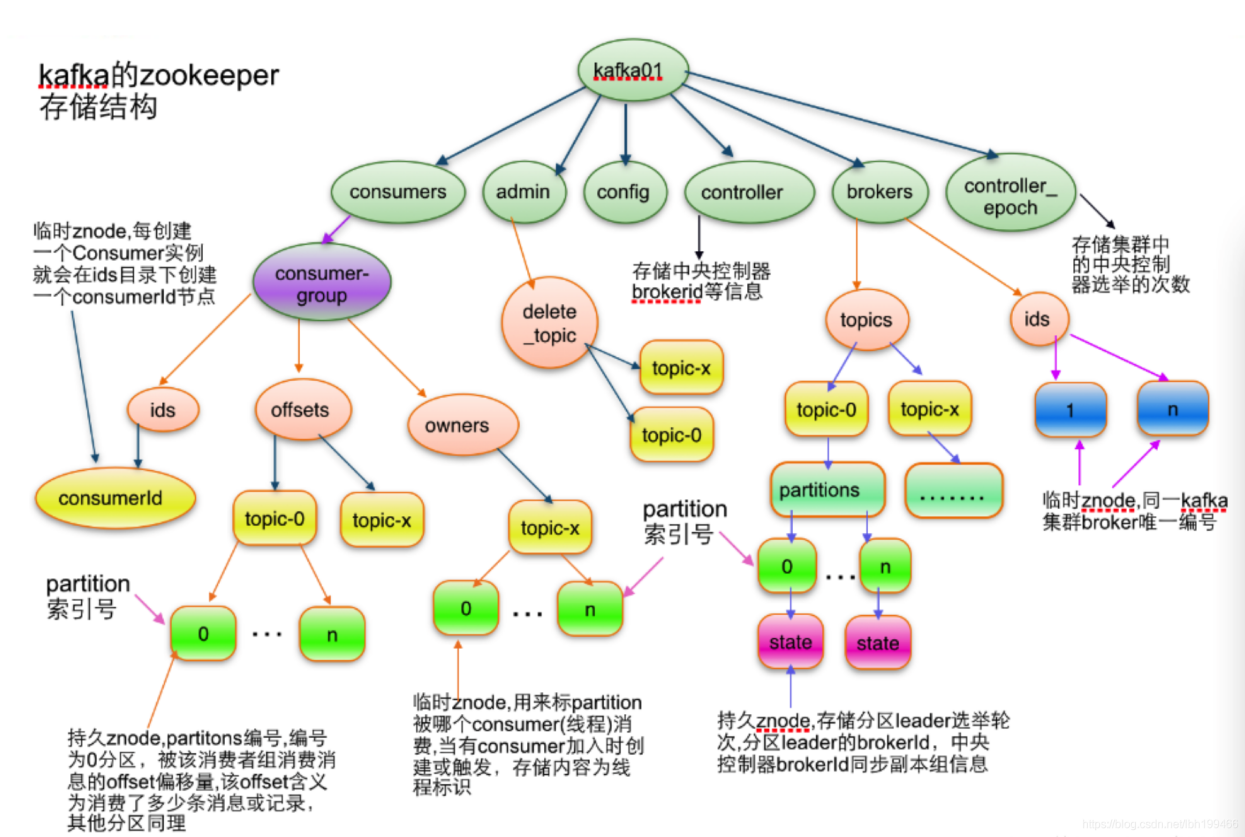
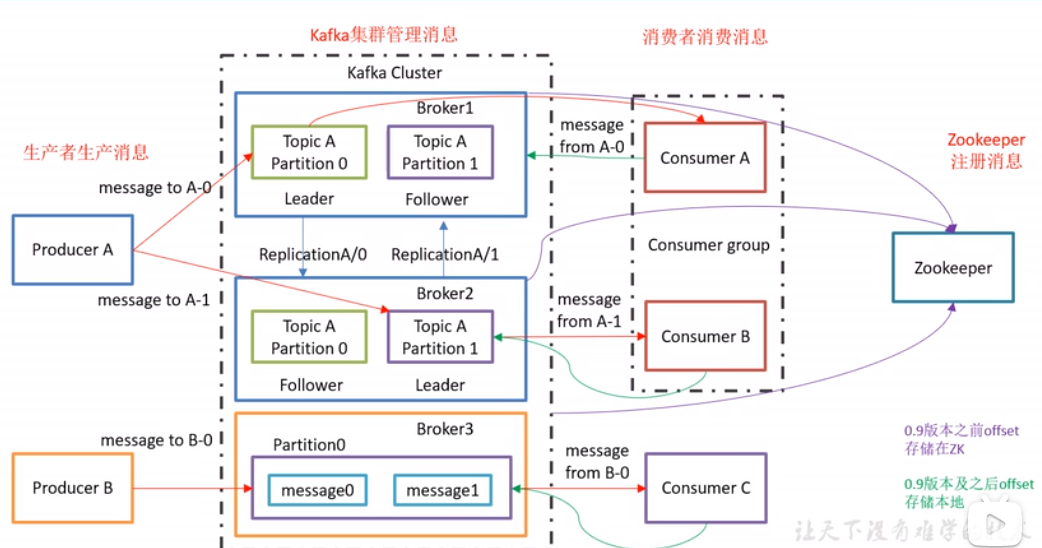
# 基础架构





Producer把消息推送到kafka的Topic中，consumer根据offset在kafka的Topic中去获取

## Producer

## Consumer

\_consumer\_offsets这个系统自定义的topic中存储了各个消费者对partition的offset

### Consumer和Partition

例如：

Topic A下有两个分区p1和p2

ConsumerGroup G1下有两个消费者 C1-1、C1-2

ConsumerGroup G2下有两个消费者 C2-1、C2-2

分析：

1. 一个partition只能被同一个消费者组里的一个消费者消费

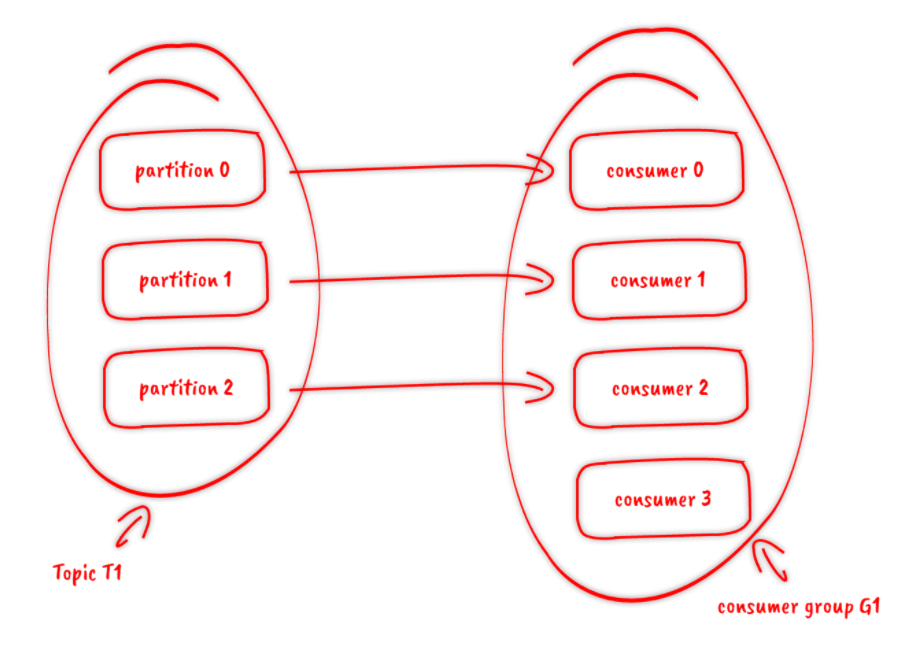
p1可以被G1和G2中的消费者消费，但是对于G1来说，C1-1或者C1-2只能有一个可以消费p1，G2同理

1. 一个consumer可以消费多个partition

假设1中是C1-1消费了p1，那么C1-1还可以消费p2，但是C1-2就不能消费p1和p2了，应该C1-2和C1-1在同一个消费者组中

所以

**所以，如果消费者组里的消费者数量比topic的partition数量多，那么该组中就有消费者是空闲的。**



## Consumer Group

## Broker

### 存储数据

Producer的数据时存在磁盘上的，kafka会单独开辟一块磁盘空间，**顺序写入数据**（比随机写入效率高）

### 删除策略

无论消息是否被消费，kafka都会保存所有的消息。对旧数据的删除策略有：

1. 基于时间，默认配置是168小时（7天）
2. 基于大小，默认配置是1073741824

由于kafka读取特定消息的时间复杂度是O(1)，所有删除过期数据不会提高kafka的读取效率。

## Topic

## Partition

Topic往Partition里面放数据是**轮询**着放的

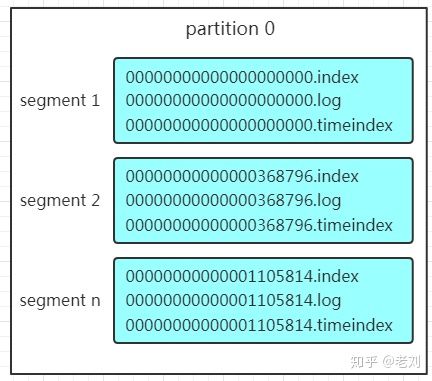
同一个分区内的数据是有序的

分区的目的：方便扩展，提高并发

1. producer写数据到kafka时可以指定partition
2. 如果没有指定partition，但是设置了数据的key，会根据key的值hash得到一个partition
3. 既没有指定partition，也没有设置数据的key，会轮询选出一个partition

### Partition结构

Partition在服务器上的表现形式就是配置文件log.dir路径下的文件夹，命名规则是{TOPIC}-{PARTITION}。每个partition文件夹下会有多组segment文件，每组segment文件包括.index文件、.log文件、.timeindex文件三个，.log文件是实际存储message数据的地方，index和timeindex文件时索引文件，用于检索消息。它们的命名方式是：以该segment最小的offset命名，下图中000.log存储offset为0~368795的消息。



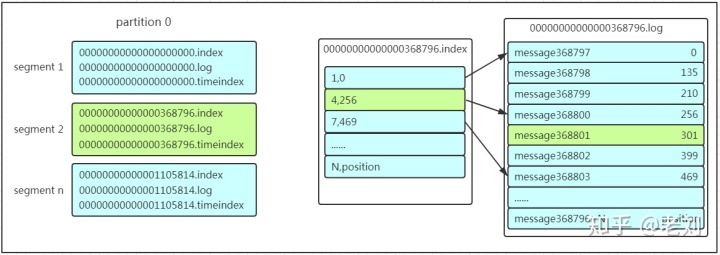
### 如何根据offset查找数据

**例子**：查找offset为368801的message

**分析**：

1. 二分查找offset为368801的message所在的segment文件
2. 打开该segment的.index文件（368796.index），计算offset为368801的message在.index文件中的**相对偏移量**为5（368796-368801）
3. 二分查找相对偏移量小于或者等于5的最大的那个相对偏移量是4，（因为.index文件采用**稀疏索引**，所以直接找相对偏移量为5的索引可能找不到）
4. 根据最大相对偏移量4找到相应索引，通过该索引确定它的message存储的物理偏移位置为256
5. 打开数据文件，从位置256那个地方开始顺序扫描直到找到offset为368801的目标数据

**总结**：使用**二分查找segment**+**有序offset**+**稀疏索引**+**二分查找偏移量**+**顺序查找**来查找指定offset的消息



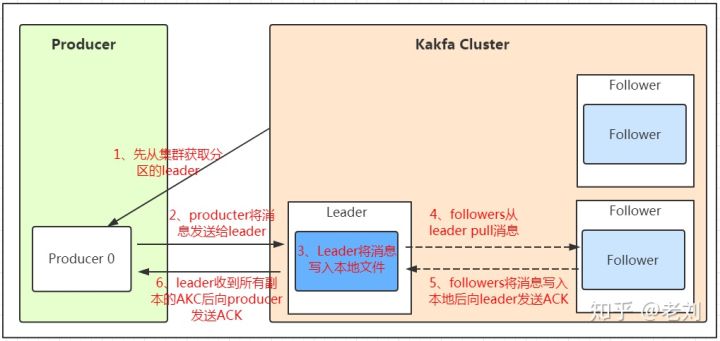
## Replication

Replication数不能大于Broker数，即leader和follower不能在一台机器上

## Leader

## Follower

Follower和leader之间同步数据



## Message

### Message结构

主要包括：

1. offset：占8字节的有序ID序号，标记自身在当前partition中的位置
2. 消息大小：占4字节，描述自身的大小
3. 消息体：存放实际的消息（被压缩过）

## 生产者

### ISR（In-Sync Replica，同步中的副本）

Kafka的partition的**所有副本叫AR**（Assigned Replica），**同步中的副本叫ISR**（In-Sync Replica），**不在同步中的副本叫OSR**（Out-Sync Replica），所以有**AR=ISR+OSR。**

Kafka的partition有一个leader和0到多个follower，leader和follower之间需要保持数据一致。这个过程不是完全同步，也不是完全异步，而是一种特殊的ISR。

Kafka需要“全部”副本都同步之后才会给producer发送ack，但是对“全部”进行了优化，动态维护一个ISR，只需要ISR中的副本同步就可以了。

Leader维护了动态的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。如果某台follower因为网络原因或者其他原因长时间未向leader同步数据，则该follower将被踢出ISR，该时间阈值有replica.lag.time.max.ms参数设定。Leader发送故障之后，将从ISR中选举新的Leader。

### 保证消息不丢失

#### Producer写入时：ACK应答机制

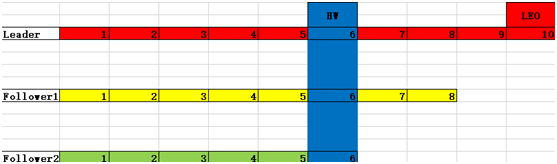
Producer向kafka写数据时，可以设置参数来确定是否确认kafka接收到的数据，这个参数可设置的值为0、1、all

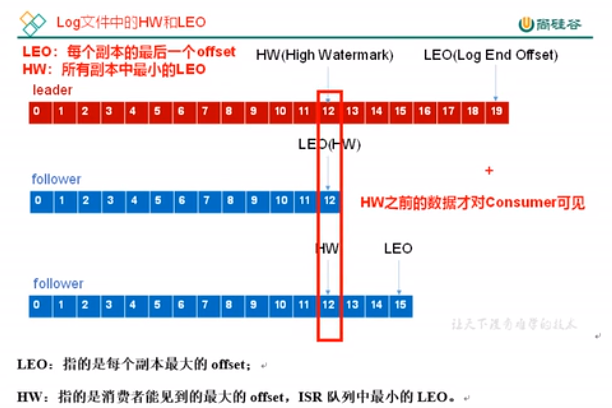
设为0：表示producer往集群写数据时不需要返回，不确保消息发送成功。安全性最低但是效率最高

设为1：表示producer往集群写数据时只要leader应答就可以发送下一条。只确保leader发送成功

设为all：表示Producer往集群发送数据时需要该leader所有的follower都完成leader的同步才会发送下一条，确保leader发送成功和所有follower都完成备份。安全性最高但是效率最低。**这时：如果ISR小于min.insync.replicas指定的数目，那么就会返回不可用**

### 保证Leader挂掉后消费数据的一致性





#### LEO（Log End Offset）

LEO是每个follower（包括leader）的下一条待写入消息的offset

#### HW（高水位）

HW是消费者能见到的最大的offset，**LSR队列中最小的LEO**

如果**采用了ack=0或者1**，所以leader和follower的offset可能会出现不一致，在这种情况下，**避免leader挂掉**后，比如当前消费者消费8这条数据后，leader挂掉了，此时follower2变成了leader，但是follower2中根本没有9这条数据，那么这是消费者就会报错，所以设计了HW这个参数，**只暴露最少的数据给消费者**，避免了上面的问题。

**TIP：leader和follower之间offset不一致是由于采用了非同步的acks机制造成的。**

#### 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。

#### Leader故障

Leader故障之后，会从ISR中选出一个新的leader，之后为了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的部分截取掉，然后从新的leader同步数据。

**这只是保证同一个分区的各个副本之间数据的一致，不保证数据不丢失或重复。**

### Exactly Once

#### At Least Once

ACK设置为-1，表示At Least Once，最少发一次，保证数据不丢失，但是不能保证数据不重复

#### At Most Once

ACK设置为0，表示At Most Once，最多发一次，保证数据不重复，但是不能保证数据不丢失

#### 幂等性

引入幂等性，确保不论Producer向Server发送多少次重复数据，Server端都只会持久化一条。幂等性结合At Least Once就是Exactly Once：

At Least Once + 幂等性 = Exactly Once

开启幂等性，只需将producer参数中的enable.idompotence设置成true即可。开启幂等性的Producer在初始化的时候会被分配一个PID，**发往同一个Partition的消息**会附带Sequence Number。而Broker端会对<PID，Partition，SeqNumber>做缓存，当具有相同主键的消息提交时，Broker只会持久化一条。？？Producer不是业务代码吗，怎么生成seqNumber？？

但是PID在重启之后会变化，同时不同的Partition也具有不同主键，所以幂等性无法保证跨分区跨会话的Exactly Once。

## 消费者

### 防止空查询

消费者采用pull拉取模式，不足之处在于kafka没有数据时，消费者会陷入空查询。针对这一点，kafka的消费者在消费数据时会传入一个市场参数timeout，如果当前没有数据可供消费，consumer会等待一段时间再返回，这段时长即为timeout。

### 分区分配策略

因为一个partition只能被同一个消费者组中的一个消费者消费，所以对于消费者，存在partition的分配问题，即partition由消费者组中的哪一个消费者进行消费。

**分配时机：消费者数量发生变化的时候**

#### Round Robin（轮询）

同一个消费者组订阅的**所有TOPIC的所有Partition**会进行hash，然后**作为一个整体以轮询**的方式由该消费者组中的消费者进行消费。

优点：同一个消费者组中的消费者分配到的partition**数量比较均衡**

但是需要满足条件：**同一个消费者组里面的消费者订阅的TOPIC需要一致**

**例如：**Consumer Group 下有C1、C2两个消费者，它们都订阅了T1、T2两个主题，T1、T2下分别有3个分区：P1-1、P1-2、P1-3、P2-1、P2-2、P2-3。

**问题：**这时6个分区P1-1、P1-2、P1-3、P2-1、P2-2、P2-3哈希之后再轮询地分配给C1、C2，最终可能C1消费P1-3、P2-2、P1-1，C2消费P1-2、P2-3、P2-1

#### Range（默认用这个）

单个TOPIC对同一个消费者组中的各个消费者进行单独的分配

可能造成的问题：数据倾斜

**例如1：**Consumer Group 下有C1、C2两个消费者，它们都订阅了T1、T2两个主题，T1、T2下分别有3个分区：P1-1、P1-2、P1-3、P2-1、P2-2、P2-3。

**问题：**这时P1-1、P1-2和P2-1、P2-2分配给了C1，P1-3、P2-3分配给了C2，这样C1就比C2多分配了两个，造成了数据倾斜

**例如2**：有CG1、CG2，CG1下有两个消费者C1、C2，CG2下有一个消费者C3，有TOPIC1、TOPIC2，TOPIC1下有P1-0、P1-1、P1-2，TOPIC2下有P2-0、P2-1、P2-2，这时C1、C2订阅了TOPIC1、TOPIC2，C3订阅了TOPIC1，这时的分区分配是：TOPIC1下的三个分区会均匀分配给CG1下的C1、C2，TOPIC2同1；TOPIC1下的三个分区会全部分配给CG2下的C3（因为CG2只有一个C3）

### offset的维护

**说明：**TOPIC A有三个partition P1、P2、P3，Consumer Group G1 现在有一个消费者C1，这时C1同时消费P1、P2、P3，offset分别为offset1、offset2、**offset3**，某时刻G1新增了一个消费者C2，这时P3重新分配，变成了C1继续消费P1、P2，C2开始消费P3。问C2是从头开始消费P3还是接着从offset3开始消费？

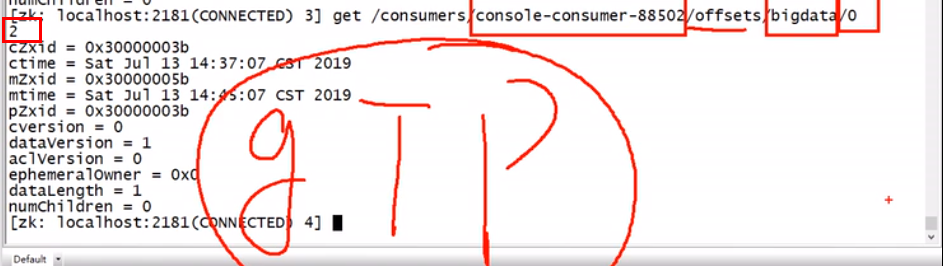
**答案：**是接着从offset3开始消费

**结论：**kafka使用**<Group, Topic, Partition>**来确定一条**offset**

#### Zk中（旧版本）

Zookeeper中的路径：

/consumer/console-consumer-{随机分配的分组ID}/offsets/{topic}/{partition}存储的就是<Group, Topic, Partition>标识的offset



#### 内置Topic（\_consumer\_offsets）

1. 修改配置文件**consumer.properties**

**exclude.internal.topics=false**

1. 命令行读取offset

/bin/kafka-console-consumer.sh **--topic \_consumer\_offsets** --zookeeper ... --formatter ...

**--consumer.config config/consumer.properties** --from-beginning

#### 自动提交和手动提交

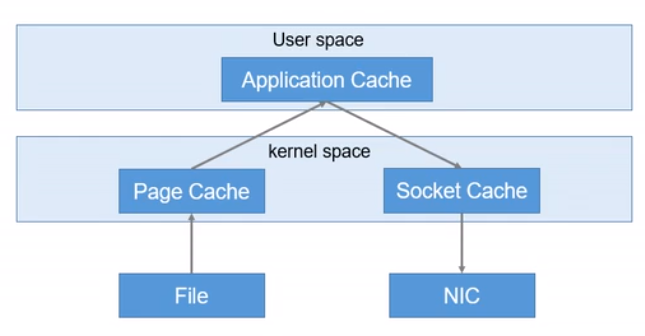
## Kafka高效读写

### 顺序写磁盘

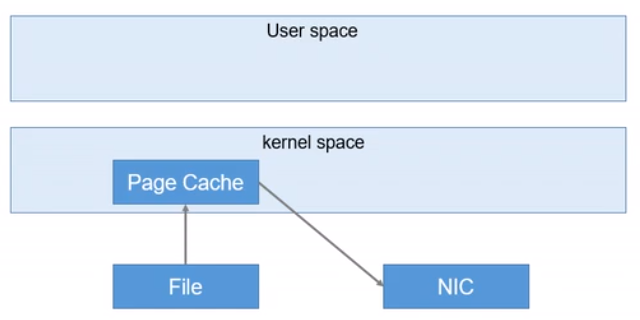
Kafka的producer生产数据，要写入到log文件中，写入的过程是一直追加到文件末端，为顺序写。顺序写的效率非常高，这与磁盘的机械结构有关，因为顺序写省去了大量磁头寻址时间。

### 零复制技术

非零复制：



零复制



## 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。

## 事务

## Rebalance

## 生产者API

### 消息发送流程

Producer发送数据采用**异步发送**（异步发送和ack机制不是一回事，ack是保证数据不丢失用的，异步是发送完一个数据之后不等响应结果就直接发第二个，但是当响应结果没有的时候可能会重发）

两个线程：**main**线程、**Sender**线程

一个共享变量：RecordAccumulator

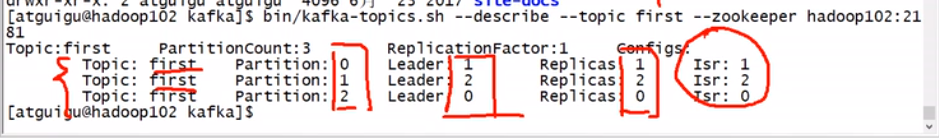
## 消费者API

# 配置文件

log.dirs：就是数据路径，使用文件夹名为{topic}:{partition}形式存在

# 命令行

1. describe显示topic信息
2. Partition后面的0,1,2是分区自己的编号。Leader和Replicas后面的0,1,2表示的是节点的broker.id。是不一样的。



1. partitions 数量可以大于broker数量，replication-factor数量不能大于broker数量
2. bin/kafka-console-consumer.sh **[--bootstrap-server]** localhost:9092 --topic test --from-beginning --partition 0

1)如果选用**--bootstrap-server**，则消费者组对每个topic的每个partition的offset数据存在kafka自身的\_consumer\_offsets中

2)如果选用**--zookeeper**，则消费者组对每个topic的每个partition的offset数据存在zookeeper的路劲下：/consumer/console-consumer-{随机分配的分组ID}/offsets/{topic}/{partition}

# Springboot 整合kafka

## 生产者

普通生产者

带回调的生产者

## 消费者