1.Kafka 中的 ISR(InSyncRepli)、OSR(OutSyncRepli)、AR(AllRepli)代表什么？

ISR+OSR=AR

ISR(InSyncRepli)：内部副本同步队列，这是副本列表的一个子集，它当前是活动的，并被提交给leader。“leader”是负责给定分区的所有读写操作的节点。每个节点将是分区中随机选择的一部分的leader，“replicas”是复制这个分区的日志的节点列表，不管它们是主节点还是活动节点。
OSR(OutSyncRepli)：外部副本同步队列
AR(AllRepli)：所有副本
ISR由leader维护，follower从leader同步数据有一些延迟（包括延迟时间replica.lag.time.max.ms和延迟条数replica.lag.max.messages两个维度，0.10.x中只支持replica.lag.time.max.ms这个维度），任意一个超过阈值都会把follower剔除出ISR, 存入OSR（OutSyncRepli）列表，新加入的follower也会先存放在OSR中
AR = ISR + OSR

2.Kafka 中的 HW、LEO 等分别代表什么？

HW：所有副本的最小的LEO，High Watermark 高水位，取一个partition对应的ISR中最小的LEO作为HW，consumer最多只能消费到HW所在的位置上一条信息。
LEO：每个副本的最后一个offset
HW/LEO这两个都是指最后一条的下一条的位置而不是指最后一条的位置
LSO：Last Stable Offset 对未完成的事务而言，LSO 的值等于事务中第一条消息的位置(firstUnstableOffset)，对已完成的事务而言，它的值同 HW 相同
LW：Low Watermark 低水位, 代表 AR 集合中最小的 logStartOffset 值

3.Kafka 中是怎么体现消息顺序性的？

kafka中的broker的每个topic的partition中的消息在写入时都是有序的，消费时，每个partition只能被每一个CG(consumer group)中的一个消费者消费，保证了消费时也是有序的。整个topic不保证有序。如果为了保证topic整个有序，那么将partition调整为1

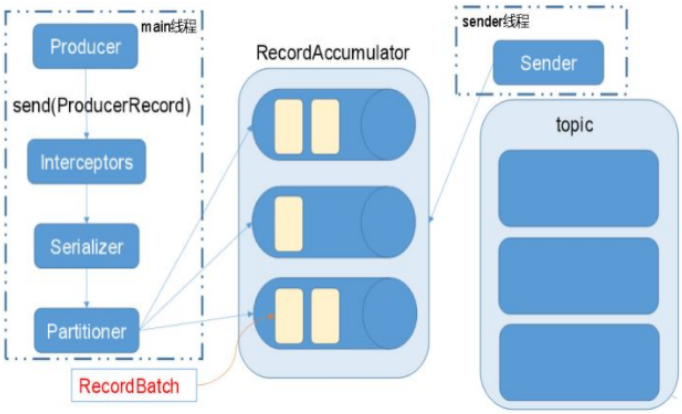
4.Kafka 中的分区器、序列化器、拦截器是否了解？它们之间的处理顺序是什么？

分区器->序列化器->拦截器

5.Kafka 生产者客户端的整体结构是什么样子的？使用了几个线程来处理？分别是什么？

答:

整体结构如图:



2个，main线程和Sender线程。main线程负责创建消息，然后通过分区器、序列化器、拦截器作用之后缓存到累加器RecordAccumulator中。Sender线程负责将RecordAccumulator中消息发送到kafka中.

6.“消费组中的消费者个数如果超过 topic 的分区，那么就会有消费者消费不到数据”这句话是否正确？

答：
为了保证数据消费的有序性，一个消费者实例只能消费一个Partition，所以如果消费者实例多了，那么会出现消费者空闲的情况。如果是自定义分区，可以继承AbstractPartitionAssignor实现自定义消费策略，从而实现同一消费组内的任意消费者都可以消费订阅主题的所有分区。

7.消费者提交消费位移时提交的是当前消费到的最新消息的 offset 还是 offset+1？

offset+1

8.有哪些情形会造成重复消费？

消费者消费后没有提交offset(程序崩溃/强行kill/消费耗时/自动提交偏移情况下unsubscrible)

9.那些情景会造成消息漏消费？

消费者没有处理完消息就提交offset(自动提交偏移 未处理情况下程序异常结束)

10.当你使用 kafka-topics.sh 创建（删除）了一个 topic 之后，Kafka 背后会执行什么逻辑？

1）会在 zookeeper 中的/brokers/topics 节点下创建一个新的 topic 节点，如：

/brokers/topics/first

2）触发 Controller 的监听程序

3）kafka Controller 负责 topic 的创建工作，并更新 metadata cache

创建：在zk上/brokers/topics/下创建一个新的topic节点，然后触发Controller的监听程序，kafkabroker会监听节点变化创建topic，kafka Controller 负责topic的创建工作，并更新 metadata cache
删除：调用脚本删除topic会在zk上将topic设置待删除标志，kafka后台有定时的线程会扫描所有需要删除的topic进行删除，也可以设置一个配置server.properties的delete.topic.enable=true直接删除

11.topic 的分区数可不可以增加？如果可以怎么增加？如果不可以，那又是为什么？

可以，例如

```
bin/kafka-topics.sh --zookeeper localhost:2181/kafka --alter --
topic topic-config --partitions 3
```

12.topic 的分区数可不可以减少？如果可以怎么减少？如果不可以，那又是为什么？

不可以，被删除的分区数据难以处理

13.Kafka 有内部的 topic 吗？如果有是什么？有什么所用？

有 __consumer Offsets 给普通消费者保存offsets用的

14.Kafka 分区分配的概念？

Range Raneroben

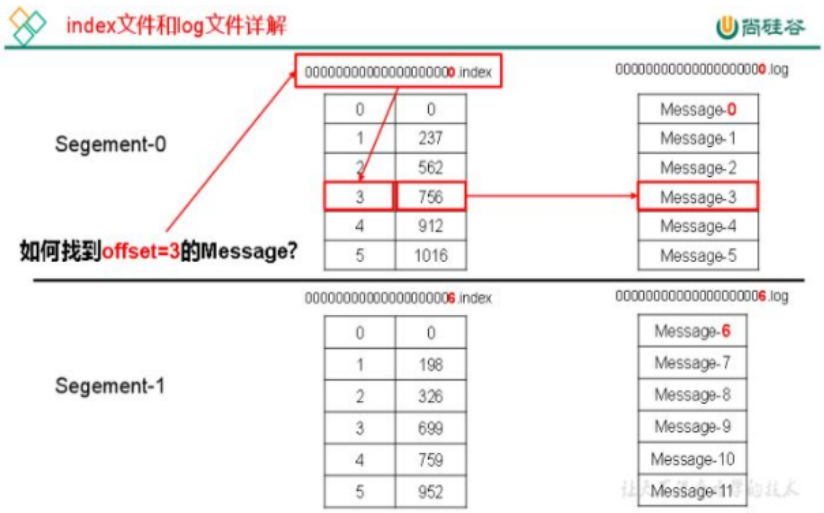
一个topic由多个分区组成，一个消费者组有多个消费者，故需要将分区分配给消费者，即确定哪个partition由哪个consumer来消费
roundrobin、range两种分配方式
round-robin：第一次调用时随机生成一个整数（后面每次调用在这个整数上自增），将这个值与topic可用的partition总数取余得到partition值
range

15.简述 Kafka 的日志目录结构？

每个partition一个文件夹，包含四类文件.index .log .timeindex leader-epoch-checkpoint
.index .log .timeindex 三个文件成对出现 前缀为上一个segment的最后一个消息的偏移
log文件中保存了所有的消息
index文件中保存了稀疏的相对偏移的索引
timeindex保存的则是时间索引
leader-epoch-checkpoint中保存了每一任leader开始写入消息时的offset，会定时更新
follower被选为leader时会根据这个确定哪些消息可用

16.如果我指定了一个 offset，Kafka Controller 怎么查找到对应的消息？

1. 通过文件名前缀数字x找到该绝对offset 对应消息所在文件
2. offset-x为在文件中的相对偏移
3. 通过index文件中记录的索引找到最近的消息的位置
4. 从最近位置开始逐条寻找
   如图：



17.聊一聊 Kafka Controller 的作用？

18.Kafka 中有那些地方需要选举？这些地方的选举策略又有哪些？

Controller：抢资源选举　leader：ISR---的选举机制

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。

19.失效副本是指什么？有那些应对措施？

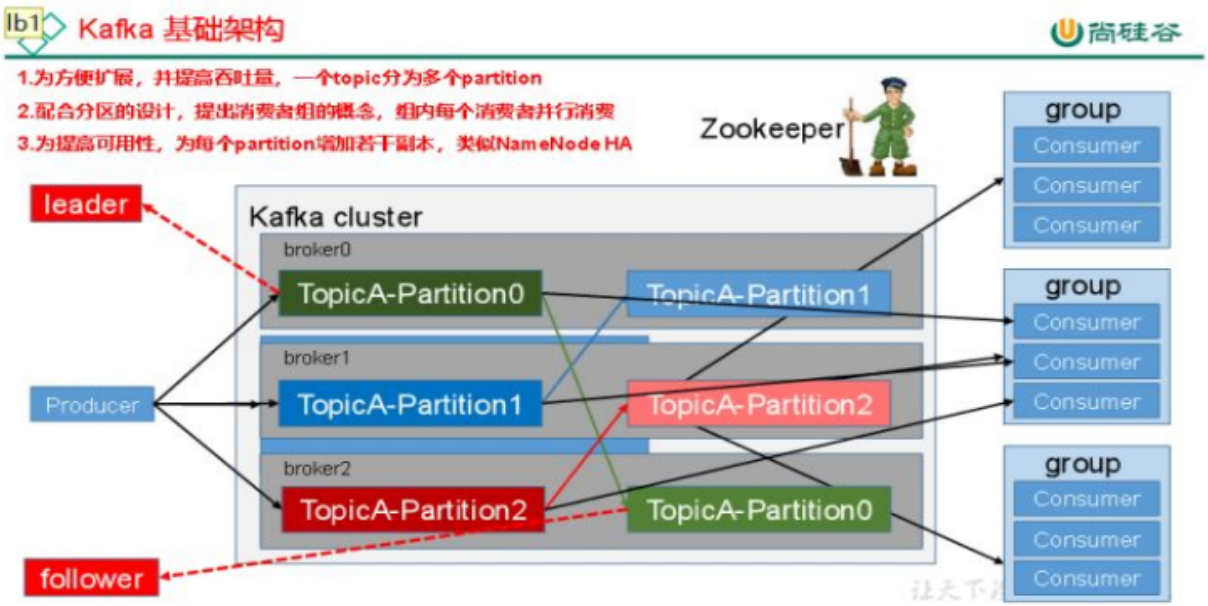
不能及时与leader同步，暂时踢出ISR，等其追上leader之后再重新加入

20.Kafka 的哪些设计让它有如此高的性能？

分布式 顺序写磁盘 零拷贝

分区
Cache Filesystem Cache PageCache缓存
顺序写磁盘，由于现代的操作系统提供了预读和写技术，磁盘的顺序写大多数情况下比随机写内存还要快。
Batching of Messages 批量量处理。合并小的请求，然后以流的方式进行交互，直顶网络上限
Pull 拉模式 使用拉模式进行消息的获取消费，与消费端处理能力相符。
0-copy 零拷技术减少拷贝次数

21. kafka基础架构



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

Consumer ：消息消费者，向kafka broker取消息的客户端；

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。

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。

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；

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；

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。

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。

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follower。

22. 如果leader故障时，ISR为空怎么办
    答：
    kafka在Broker端提供了一个配置参数：unclean.leader.election,这个参数有两个值：
    true（默认）：允许不同步副本成为leader，由于不同步副本的消息较为滞后，此时成为leader，可能会出现消息不一致的情况。
    false：不允许不同步副本成为leader，此时如果发生ISR列表为空，会一直等待旧leader恢复，降低了可用性。
23. kafka的message格式是什么样的
    答：
    一个Kafka的Message由一个固定长度的header和一个变长的消息体body组成
    header部分由一个字节的magic(文件格式)和四个字节的CRC32(用于判断body消息体是否正常)构成。
    当magic的值为1的时候，会在magic和CRC32之间多一个字节的数据：
    attributes(保存一些相关属性，比如是否压缩、压缩格式等等);如果magic的值为0，那么不存在attributes属性
    body是由N个字节构成的一个消息体，包含了具体的key/value消息
24. kafka中consumer group 是什么概念
    答：
    和topic一样，是逻辑上的概念，是Kafka实现单播和广播两种消息模型的手段。同一个topic的数据，会广播给不同的group；同一个group中的consumer，只有一个consumer能拿到这个数据。换句话说，对于同一个topic，每个group都可以拿到同样的所有数据，但是数据进入group后只能被其中的一个consumer消费。group内的consumer可以使用多线程或多进程来实现，也可以将进程分散在多台机器上，consumer的数量通常不超过partition的数量，且二者最好保持整数倍关系，因为Kafka在设计时假定了一个partition只能被一个consumer消费（同一group内）。
25. 为什么Kafka不支持读写分离？
    答：
    在 Kafka 中，生产者写入消息、消费者读取消息的操作都是与 leader 副本进行交互的，从 而实现的是一种主写主读的生产消费模型。

Kafka 并不支持主写从读，因为主写从读有两个很明显的缺点:

数据一致性问题。数据从主节点转到从节点必然会有一个延时的时间窗口，这个时间 窗口会导致主从节点之间的数据不一致。某一时刻，在主节点和从节点中 A 数据的值都为 X，之后将主节点中 A 的值修改为 Y，那么在这个变更通知到从节点之前，应用读取从节点中的 A 数据的值并不为最新的 Y，由此便产生了数据不一致的问题。

延时问题。类似 Redis 这种组件，数据从写入主节点到同步至从节点中的过程需要经 历网络→主节点内存→网络→从节点内存这几个阶段，整个过程会耗费一定的时间。而在 Kafka 中，主从同步会比 Redis 更加耗时，它需要经历网络→主节点内存→主节点磁盘→网络→从节 点内存→从节点磁盘这几个阶段。对延时敏感的应用而言，主写从读的功能并不太适用。

26. kafka如何实现延迟队列？

    Kafka并没有使用JDK自带的Timer或者DelayQueue来实现延迟的功能，而是基于时间轮自定义了一个用于实现延迟功能的定时器（SystemTimer）。JDK的Timer和DelayQueue插入和删除操作的平均时间复杂度为O(nlog(n))，并不能满足Kafka的高性能要求，而基于时间轮可以将插入和删除操作的时间复杂度都降为O(1)。时间轮的应用并非Kafka独有，其应用场景还有很多，在Netty、Akka、Quartz、Zookeeper等组件中都存在时间轮的踪影。

底层使用数组实现，数组中的每个元素可以存放一个TimerTaskList对象。TimerTaskList是一个环形双向链表，在其中的链表项TimerTaskEntry中封装了真正的定时任务TimerTask.

Kafka中到底是怎么推进时间的呢？Kafka中的定时器借助了JDK中的DelayQueue来协助推进时间轮。具体做法是对于每个使用到的TimerTaskList都会加入到DelayQueue中。Kafka中的TimingWheel专门用来执行插入和删除TimerTaskEntry的操作，而DelayQueue专门负责时间推进的任务。再试想一下，DelayQueue中的第一个超时任务列表的expiration为200ms，第二个超时任务为840ms，这里获取DelayQueue的队头只需要O(1)的时间复杂度。如果采用每秒定时推进，那么获取到第一个超时的任务列表时执行的200次推进中有199次属于“空推进”，而获取到第二个超时任务时有需要执行639次“空推进”，这样会无故空耗机器的性能资源，这里采用DelayQueue来辅助以少量空间换时间，从而做到了“精准推进”。Kafka中的定时器真可谓是“知人善用”，用TimingWheel做最擅长的任务添加和删除操作，而用DelayQueue做最擅长的时间推进工作，相辅相成。