| 序号 | 问题 | 答案 |
| --- | --- | --- |
| 1 | 什么是ISR？如何通过ISR保证partition的高可用性？ | ISR（In-Sync Replicas）副本同步队列。leader维护ISR列表。follower从leader同步数据，如果出现延迟，会将follower踢出ISR到OSR中（Outof-Sync Replicas）。follower在之后追上leader后，重新加入到ISR。对于leader新的消息，其他follow用复制，当所有ISR中的follower都同步消息，leader会改变HW位置。HW位置决定comsumer可以pull到哪些消息。所以就算leader挂掉，会在ISR中选举新的leader来。 |
| 2 | 阐述consumer与partition的消费对应关系？（分析consumer增加／减少，partition增加／减少等场景下的消费再均衡） | a. 消费组增加组员，最多增加到和partition数量一致，超过的组员只会占用资源，而起不到作业。  b. partition个数，一定要大于消费组组员的个数，并且partition的个数对于消费组组员取模为0，要不有些消费者会占用资源不起作用 |
| 3 | kafka中消息如何分区？分区策略是怎样的？ | 创建topic的时候通过partitions参数指定分区数量，当发送的消息指定分区则放入指定分区内，没有指定分区则使用默认分区选择器， 通过DefaultPartitioner实现分区，分区的策略：key存在：hash(key) % numPartition，key不不存在：采用round-robin算法，每个分区机会均等，也可通过实现Partitioner接口配置自定义的分区选择器。 |
| 4 | 分析下kafka的消息格式，目前有三个版本(v1/v2/v3),分析下v1即可 | 每条消息定义为一个Record, 每个Record必定对应一个offset和message size。每条消息都一个offset用来标志它在partition中的偏 移量，这个offset是逻辑值，而非实际物理偏移值，message size表示消息的大小，这两者的一起被称之为日志头部 （LOG\_OVERHEAD），固定为12B； 消息（Record）格式中的各个字段的解释如下：  1.crc32（4B）：crc32校验值。校验范围为magic至value之间；  2.magic（1B）：消息格式版本号，此版本的magic值为1；  3.attributes（1B）：消息的属性。总共占1个字节，低3位表示压缩类型：0表示NONE、1表示GZIP、2表示SNAPPY、3表示LZ4（LZ4自Kafka 0.9.x引入），其余位保留；  4.timestamp：时间戳；  5.key length（4B）：表示消息的key的长度。如果为-1，则表示没有设置key，即key=null；  6.key：可选，如果没有key则无此字段；  7.value length（4B）：实际消息体的长度。如果为-1，则表示消息为空；  8.value：消息体。可以为空，比如tomnstone消息。 |
| 5 | kakfa数据落在磁盘，为啥还可以保持高速的读写？ | 首先是因为Kafka在磁盘上只做Sequence I/O，由于消息系统读写的特殊性，这并不存在什么问题；所以通过只做Sequence I/O的限 制，规避了磁盘访问速度低下对性能可能造成的影响；Kafka重度依赖底层操作系统提供的PageCache功能，当上层有写操作时，操作系统只是将数据写PageCache，同时标记Page属性为Dirty；当读操作发生时，先从PageCache中查找，如果发生缺页才进行磁盘调度，最终返回需要的数据；实际PageCache是把尽可能多的空闲内存都当做了磁盘缓存来使用。同时如果有其他进程申请内存，回收PageCache 的代价又很小，所以现代的OS都支持PageCache；使PageCache功能同时可以避免在JVM内部缓存数据，JVM为我们提供了强大的GC能 力，同时也引入了一些问题不适用与Kafka的设计；如果在Heap内管理缓存，JVM的GC线程会频繁扫描Heap空间，带来不必要的开销；如果Heap过大，执行一次Full GC对系统的可用性来说将是极大的挑战。 |
| 6 | 描述生产者端acks不同参数对消息可靠性的影响 | acks控制多少个副本必须写入消息后生产者才能认为写入成功，这个参数对消息丢失可能性有很大影响。这个参数有三种取值： acks=0：生产者把消息发送到broker即认为成功，不等待broker的处理结果。这种方式的吞吐最高，但也是最容易丢失消息的。  acks=1：生产者会在该分区的群首（leader）写入消息并返回成功后，认为消息发送成功。如果群首写入消息失败，生产者会收到错误响应并进行重试。这种方式能够一定程度避免消息丢失，但如果群首宕机时该消息没有复制到其他副本，那么该消息还是会丢失。另外，如果我们使用同步方式来发送，延迟会比前一种方式大大增加（至少增加一个网络往返时间）；如果使用异步方式，应用感知不到延迟，吞吐量则会受异步正在发送中的数量限制。  acks=all：生产者会等待所有副本成功写入该消息，这种方式是最安全的，能够保证消息不丢失，但是延迟也是最大的。 |
| 7 | 描述消费者端自动提交和手动提交实现方式 | a. 自动提交:只需设置”enable.auto.commit”，为"true"即可，"auto.commit.interval.ms",为默认自动提交的时间，一般设为"1000"，单位ms;  b. 手动提交:设置”enable.auto.commit”，为"false"。手动提交有两种形式，一种是提交所有订阅的topic，直接在每次消费完成之后调用consumer.commitSync()；另一种则是单独提交一个topic的partition，调用形式consumer.commitSync(Collections.singletonMap(partition, new OffsetAndMetadata(lastOffset + 1))，这种形式使我们对topic可以有更加精细的控制 |
| 8 | Kafka的使用场景 | 日志收集/消息系统/用户活动跟踪/流式处理 |
| 9 | 描述下Kafka文件的存储结构 | 每个part在内存中对应一个index，记录每个segment中的第一条消息偏移。  a. segment file组成：由2大部分组成，分别为index file和data file，此2个文件一一对应，成对出现，后缀".index"和“.log”分别表示为segment索引文件、数据文件.  b. segment文件命名规则：partion全局的第一个segment从0开始，后续每个segment文件名为上一个全局partion的最大offset(偏移message数)。数值最大为64位long大小，19位数字字符长度，没有数字用0填充。 |