大家好，上一节我们对Redis进行了学习，本小节中我们主要对消息队列Kafka进行简单的学习与介绍。消息队列也是服务端的通用工具之一，在众多的场景中都有使用。消息队列的了解与掌握是面试中的一大加分项。 3

（1）消息队列Kafka有了解吗？

答：Kafka是一个消息队列，可以实现发布订阅模式，在异步通信或者生产者和消费者需要解耦合的场景中经常使用，可以对数据流进行处理等。

Kafka的特性如下所示：

- Kafka支持消息的快速持久化
- 支持批量读写消息
- 支持消息分区，并且支持在线增加分区，提高了并发能力
- 支持为每个分区创建多个副本

Kafka可以**实现消息的快速持久化**的原因：

- KafKa将消息保存在磁盘中，并且读写磁盘的方式是顺序读写，避免了随机读写磁盘（寻道时间过长）导致的性能瓶颈。
- 磁盘的顺序读写速度超过内存随机读写。

解析：

这道题目是Kafka相关知识点的基础题目。如果我们的简历中写了对Kafka有一定的了解与掌握或者面试的岗位对消息队列有一定的了解，那么肯定会受到来自灵魂的拷问。
读到这里，聪明的读者可能会有疑问了。**Kafka使用磁盘进行数据的持久化，那么如何保证高性能呢？**我们都知道磁盘的操作一般情况下是远远低于对内存的操作效率的。

Kafka使用磁盘存储，为什么会具有高性能的特点？

- **顺序读写磁盘：**

消息在磁盘中的方式是顺序读写的，磁盘的顺序读写速度超过内存随机读写。

- **页缓存：**

页缓存是操作系统实现的一种主要的磁盘缓存，以此用来减少对磁盘I/O 的操作。具体就是把磁盘中的数据缓存到内存中，把对磁盘的访问变为对内存的访问。当然，也会存在磁盘脏页，以及合适时机会进行刷盘操作。

- **零拷贝：**

使用零拷贝（ Zero-Copy ）技术来进一步提升Kafka性能。零拷贝是指将数据直接从磁盘文件复制到网卡设备中，而不需要经由应用程序之手。零拷贝大大提高了应用程序的性能，减少了内核和用户模式之间的上下文切换

（2）Kafka中的核心概念：

答： 核心概念如下所示：

- **生产者（Producer）：**

生产消息，并且按照一定的规则（分区分配规则）推送到Topic的分区中。

- **消费者（Consumer）：**

从Topic中拉取消息并且进行消费，消费者自行维护消费消息的位置（offset）。 3

- **主题（Topic）：**

存储消息的逻辑概念，是一个消息集合，Kafka根据topic对消息进行归类，发布到Kafka集群的每条消息都需要指定一个topic。

- **分区（partition）：**

每个Topic可以划分为多个分区，每个消息在分区中都会有一个唯一编号offset，kafka通过offset保证消息在分区中的顺序，同一Topic的不同分区可以分配在不同的Broker上，partition以文件的形式存储在文件系统中。

- **副本（replica）：**

KafKa对消息进行了冗余备份，每个分区有多个副本，每个副本中包含的消息是“一样”的。每个副本中都会选举出一个Leader副本，其余为Follower副本，Follower副本仅仅是将数据从Leader副本拉到本地，然后同步到自己的Log中。

- **消费者组（Consumer Group）：**

每个consumer都属于一个consumer group，每条消息只能被consumer group中的一个Consumer消费，但可以被多个consumer group消费。

- **Broker：**

一个单独的server就是一个Broker，主要用来接收生产者发过来的消息，分配offset，并且保存到磁盘中。

- **Cluster & Controller：**

多个Broker可以组成一个Cluster集群，每个集群选举一个Broker来作为Controller，充当指挥中心。Controller负责管理分区的状态，管理每个分区的副本状态，监听ZooKeeper中数据的变化等工作。

- **日志压缩与保留策略：**

不管消费者是否已经消费了消息，Kafka都会保存这些消息（持久化到磁盘），通过配置相应的保留策略，定时删除陈旧的消息。所谓日志压缩，就是定时进行相同key值得合并，只保留最新的Key-Value值。

（3）简单介绍下Kafka中的副本机制吧。

答：在上边核心概念的介绍中，我们提到了使用副本机制来进行冗余备份，这里我们继续介绍副本机制的相关知识点。

在分布式的存储中，进行冗余备份是一种常见的设计，主要的设计方案有同步复制和异步复制。

同步复制：

当所有的Follower副本都将消息复制完成，这条消息才会被认为是提交完成，一旦有一个Follower副本出现故障，就会导致消息无法提交，极大的影响到了系统的性能。

异步复制：

当Leader副本接收到生产者发送的消息后就认为当前消息提交成功。Follower副本异步的从Leader副本同步消息，但是不可以保证同步速度，当Leader副本突然宕机的时候，可能Follower副本中的消息落后太多，导致消息的丢失。

考虑到同步复制和异步复制的优缺点，Kafka引入了ISR集合。

ISR（In-Sync-Replica）集合：

可用副本集合，ISR集合表示**当前"可用"**且消息量与Leader**相差不多**的副本集合，需要满足如下条件：

- 副本所在节点必须维持着与ZooKeeper的连接。
- 副本最后一条信息的offset与Leader副本的最后一条消息的offset之间的差值不能超过指定的阈值。

HW和LEO标志：

- **HW（HighWatermark）**表示高水位，标记了一个特殊的offset，当消费者处理消息的时候，只能拉取到HW之前的消息。HW也是由Leader副本管理的。
- **LEO（Log End Offset）**是所有副本都会有的一个offset标记。

ISR、HW和LEO的工作配合机制：

- producer向此分区中推送消息
- Leader副本将消息追加到Log中，并且递增其LEO
- Follower副本从Leader副本中拉取消息进行同步
- Follower副本将消息更新到本地Log中，并且递增其LEO
- 当ISR集合中的所有副本都完成了对offset的消息同步，Leader副本会递增其HW

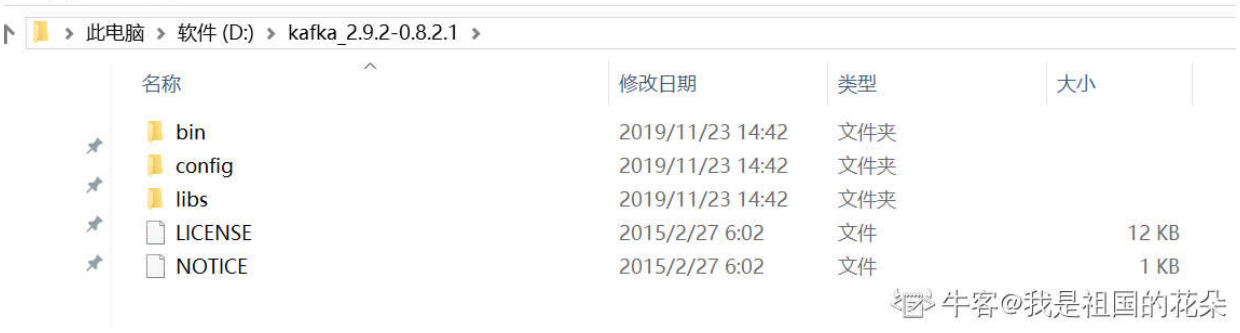
优势：

- 同步复制会导致高延迟，异步复制可能会造成消息的丢失。
- KafKa引入的ISR集合解决了同步复制和异步复制的缺点。
- 当Follower副本延迟过高时，将会被踢出ISR集合，避免了高延迟的Follower副本影响整个KafKa集群性能。
- 当Leader副本所在的Broker宕机，会优先将ISR集合中的Follower副本选举为Leader。

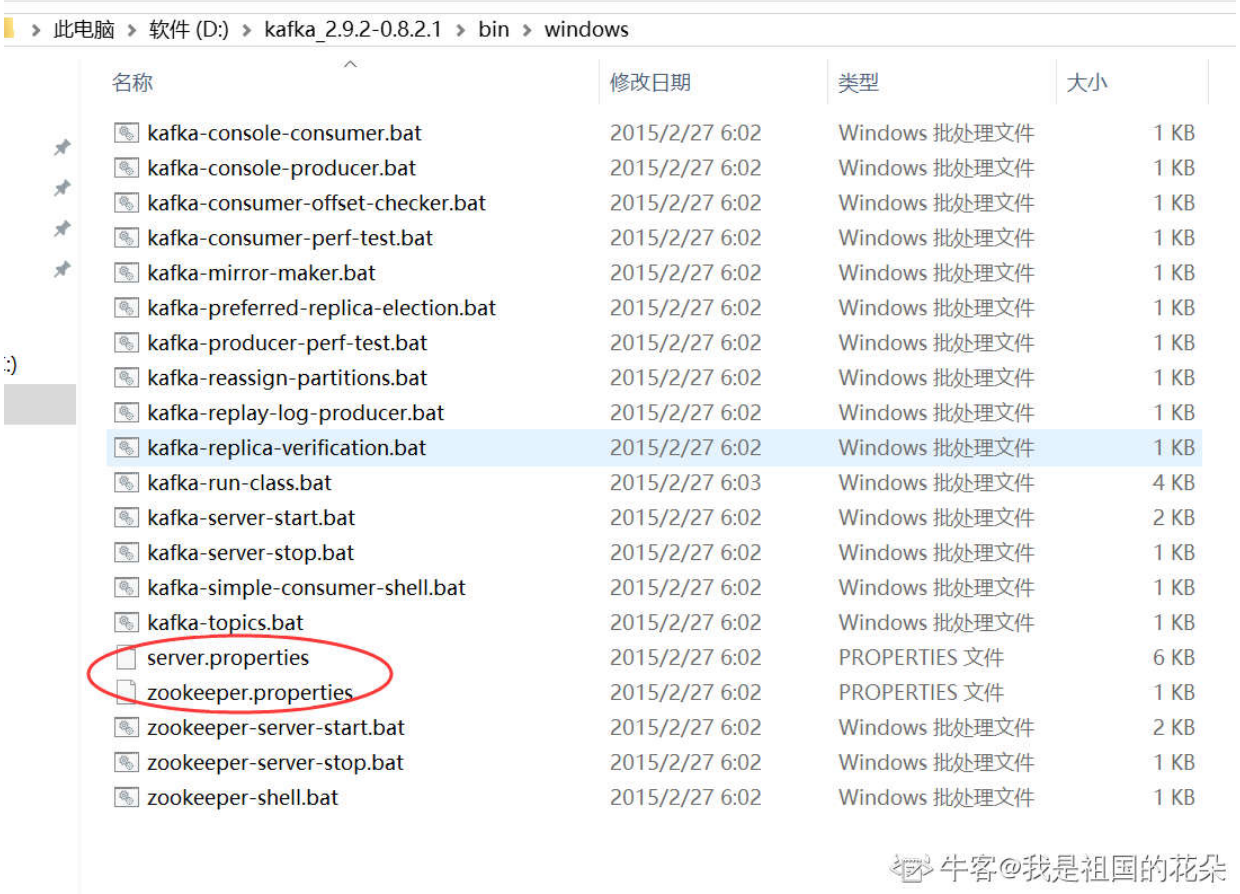
学习到这里，我们来简单搭建一个Kafka的本地调试环境，看看Kafka的效果吧。

Kafka本地调试环境

首先我们需要下载安装包：kafka_2.9.2-0.8.2.1.tgz，解压之后如图所示：

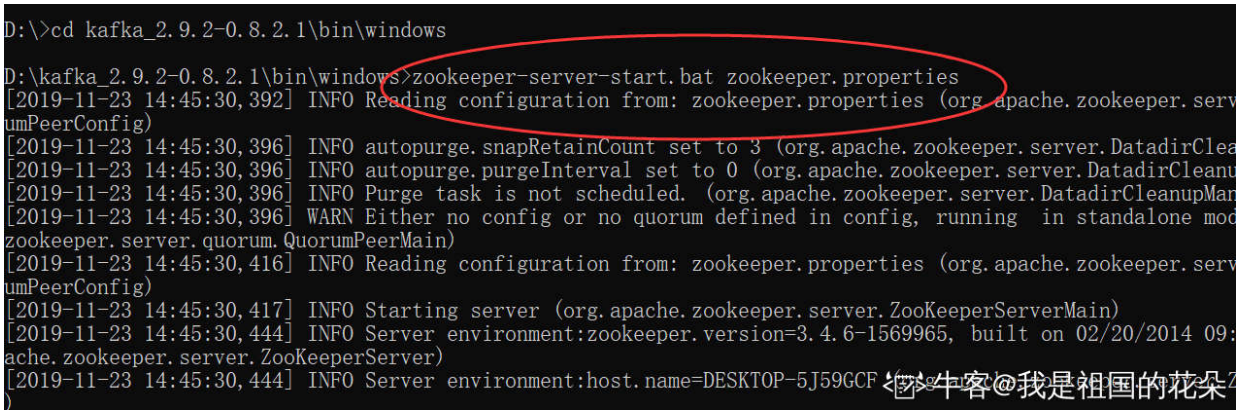


配置修改zookeeper.properties与server.properties修改为本地路径，也就是将config文件夹中的zookeeper.properties与 server.properties拷贝到bin/windows下，在bin\windows目录下存着windows的服务脚本。如图所示：



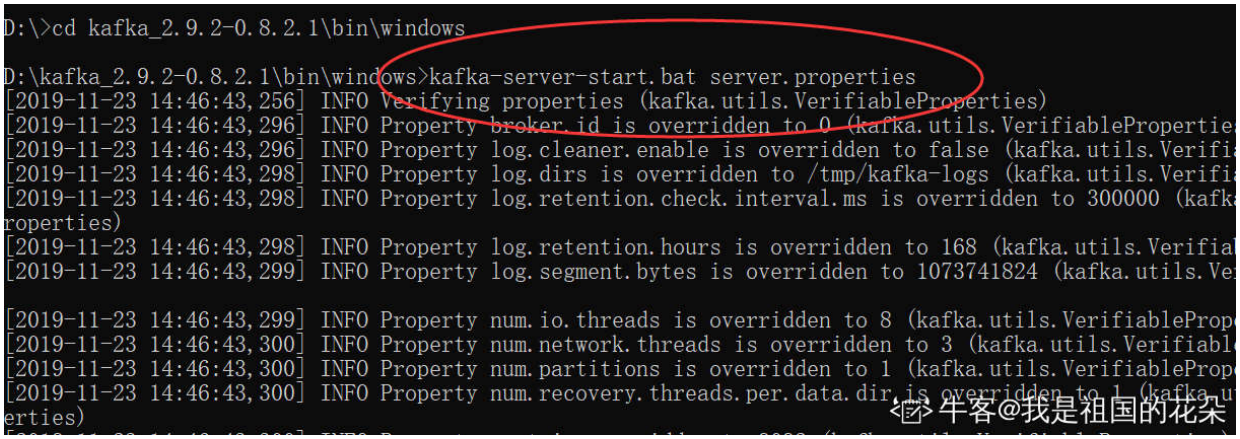
然后，我们cd到bin\windows目录，使用如下的脚本启动zookeeper服务，如下所示：

```
zookeeper-server-start.bat zookeeper.properties
```



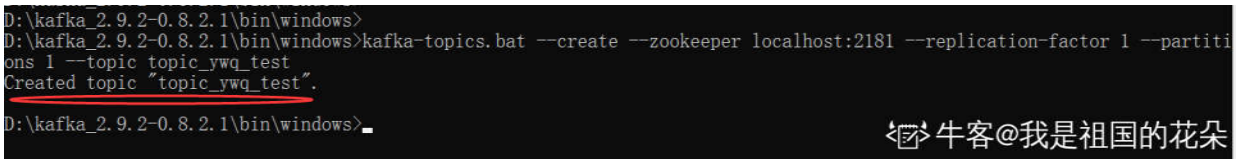
接下来，我们启动启kafka

```
kafka-server-start.bat server.properties
```



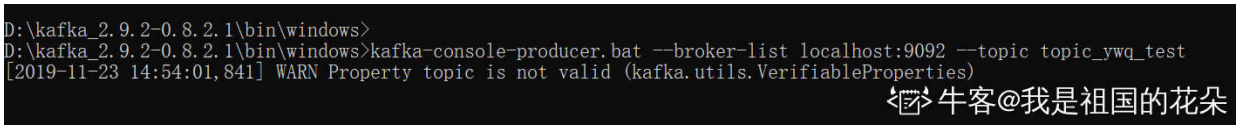
消费测试，我们建立测试使用的topic，即**topic_ywq_test**

```
kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 -
```



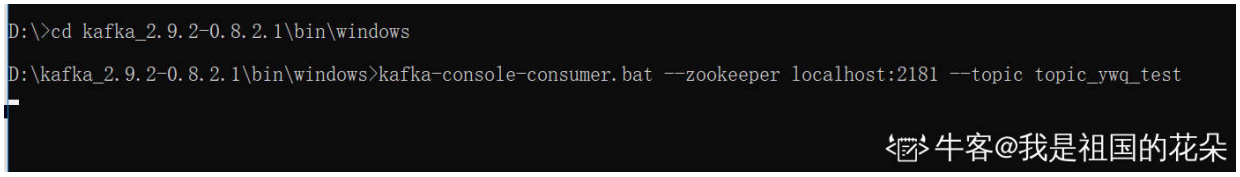
接下来，我们建立一个Producer

```
kafka-console-producer.bat --broker-list localhost:9092 --topic topic_ywq_test
```

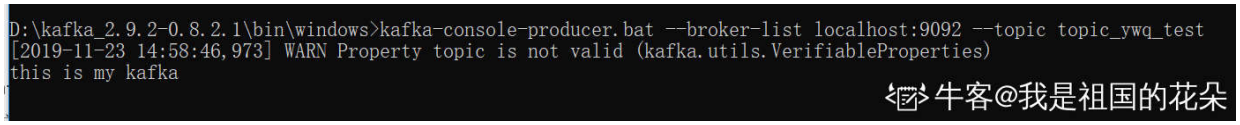


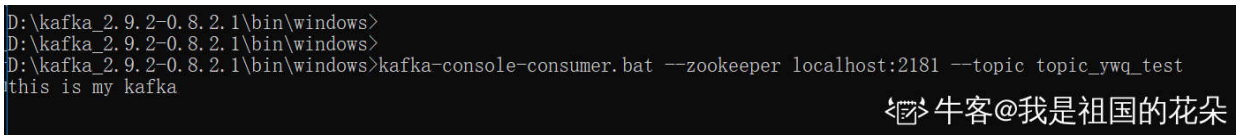
建立一个Consumer, 执行如下的命令:

```
kafka-console-consumer.bat --zookeeper localhost:2181 --topic topic_ywq_test
```



然后我们在生产者窗口输入**this is my kafka**，则消费者窗口会原样显示。





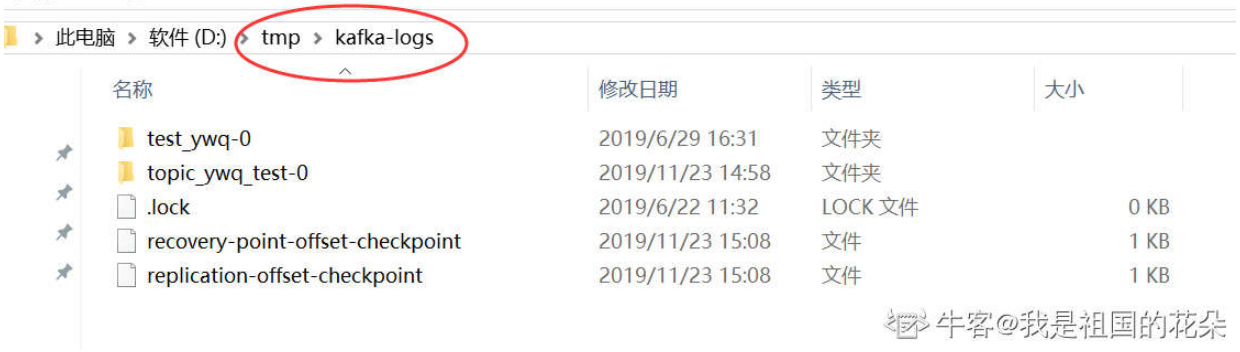
到这里，我们就搭建了一个简单的Kafka的本地调试环境，并且实现了生产者和消费者的合作，接下来，我们继续学习Kafka的相关技术原理吧。

（4）Kafka的文件存储机制：

答：Kafka中消息是以topic进行分类的，生产者通过topic向Kafka broker发送消息，消费者通过topic读取数据。

然而topic在物理层面又能以partition为分组，一个topic可以分成若干个partition，partition还可以细分为segment，一个partition物理上由多个segment组成。

在**server.properties**中可以设置文件的存储位置，默认为**log.dirs=/tmp/kafka-logs**。当我们创建一个topic的时候，可以在/tmp/kafka-logs目录中看到对应分区个数的目录数。在Kafka文件存储中，同一个topic下有多个不同的partition，每个partiton为一个目录，partition的名称规则为：topic名称+有序序号，第一个序号从0开始计，最大的序号为partition数量减1

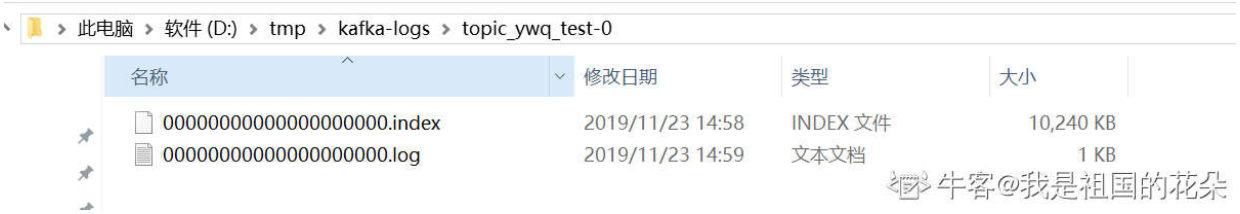


解析：

我们再来接着看下segment的介绍吧。如果以partition为最小存储单位，当Producer不断发送消息，必然会引起partition文件的无限扩张，这样不利于消息文件的维护以及已经被消费的消息的清理。

每个partition(目录)相当于一个巨型文件被平均分配到多个大小相等的segment(段)数据文件中（每个segment 文件中消息数量不一定相等）。这种特性也方便old segment的删除，即方便已被消费的消息的清理，提高磁盘的利用率，每个partition只需要支持顺序读写就行。

segment文件由两部分组成，分别为“.index”文件和“.log”文件，分别表示为segment索引文件和数据文件。



接下来，我们继续阐述Topic和Partition的相关知识点。

（5）Topic和Partition：

答：Kafka中的一个topic可以认为是一类消息，每个topic将被分成多个partition，每个partition在存储层面是append log文件。发布到此partition的消息都会被追加到log文件的尾部，**每条消息在文件中的位置称为offset(偏移量)**，offset为一个long型的数字，它唯一标记一条消息。每条消息都被append到partition中，是一种顺序写磁盘的方式，**顺序写磁盘**效率比随机写内存还要高，这也是Kafka高吞吐率的一个很重要的保证。

分区的水平扩展：

每一条消息被发送到broker中，会根据partition规则选择被存储到哪一个partition。如果partition规则设置的合理，所有消息可以均匀分布到不同的partition里。

这里需要注意如下的问题：

为什么Kafka中的分区只支持增长，不支持减小分区个数的操作?

- 删除掉的分区的消息不好处理，若丢弃则可靠性得不到保证

- 如果插入现有分区的尾部，则一些带时间戳的消息会对消费者有影响
- 如果消息量大的话，复制到其它分区也会很耗费资源；

如果你非要减小分区个数，那么可以新创建一个分区数比较小的topic，将现有topic中的消息按照一定的逻辑复制过去。

3

（6）Kafka消息传输的三大语义：

答：Kafka有以下三种可能的传输保障（delivery guarantee）：

- **At most once: 最多一次**，消息可能会丢，但绝不会重复传输
- **At least once：最少一次**，消息绝不会丢，但可能会重复传输
- **Exactly once：恰好一次**，每条消息肯定会被传输一次且仅传输一次

At most once：最多消费一次，绝对不会重复消费。**那么Kafka如何保证At most once语义呢?**

- 生产者Producer来生产消息的时候，当写数据失败的时候，broker直接跳过该消息，导致消息丢失，消费者无法消费该消息。
- 消费者在拉取消息之后，直接提交消息位移offset，但是没有完全消费完拉取消息即发生故障，下次会直接从刚刚offset的位置进行消费，刚刚故障时刻-offset之间的消息丢失。

At least once：最少一次，消息绝不会丢。那么Kafka如何保证最少消费一次呢?

生产者在生产数据的时候，以及写入了broker中，但是由于broker上的异常，导致生产者并没有成功的收到ACK，之后会进行重试操作，导致消息被写入了多次。

Exactly once：恰好消费一次，那么Kafka如何保证恰好一次的语义呢?

- **生产者生产消息的时候保证幂等性。**对于同一个数据无论操作多少次都只写入一条数据，如果重复写入，则执行不成功。
- **跨partition的原子性写操作。**broker写入数据的时候，保证原子性操作，要么写入成功，要么写入失败。（不成功不断进行重试）

（7）Kafka中关于可靠性的配置：

答：Kafka中有一些关于可靠性的配置参数，我们这里进行一个简单的介绍。

request.required.acks：

当producer向leader发送数据时，可以通过request.required.acks参数来设置数据可靠性的级别：

- **1（默认）**：这意味着producer在ISR中的leader已成功收到数据并得到确认。如果leader宕机了，则会丢失数据。
- **0**：这意味着producer无需等待来自broker的确认而继续发送下一批消息。这种情况下数据传输效率最高，但是数据可靠性确是最低的。
- **-1**：producer需要等待ISR中的所有follower都确认接收到数据后才算一次发送完成，可靠性最高。但是这样也不能保证数据不丢失，比如当ISR中只有leader时（ISR中的成员由于某些情况会增加也会减少，最少就只剩一个leader），这样就变成了acks=1的情况。

min.insync.replicas（ISR集合中的最小副本个数）：

为了提高数据的可靠性，在设置request.required.acks=-1的同时，也要设置ISR中的最小副本个数min.insync.replicas这个参数。 3

- min.insync.replicas这个参数设定ISR中的最小副本数是多少，默认值为1，当且仅当request.required.acks参数设置为-1时，此参数才生效。
- 如果ISR中的副本数少于min.insync.replicas配置的数量时，客户端会返回异常：org.apache.kafka.common.errors.NotEnoughReplicasExceptoin: Messages are rejected since there are fewer in-sync replicas than required

（8）Kafka中其余重要知识点：

答：这篇文章，我们主要针对常见的基础知识点做了一个简单的介绍与学习，有能力或者有需求的同学可以继续深入学习如下的重要知识点：

- Kafak中的消费者和消费者组
- Kafka分区分配以及持久化
- Kafka的发送模式等

总结：

本小节中，我们主要对消息队列Kafka的基本概念进行了简单的介绍，对于Kafka没有基础的同学看着可能有点费劲，但是如果我们想要叩开大厂之门，对于Kafka还是有必要进行相应的学习与了解的。

关于第三章节中的redis和Kafka，在面试中都可深可浅，在一些大公司都拥有自研的消息队列，但是其基本原理都和Kafka相似。有能力和精力的同学可以进一步深入研究学习Kafka相关技术原理。下一小节，我们将进入网络协议篇章，主要阐述基本网络协议以及网络安全的相关知识点。

限于作者水平，文章中难免会有不妥之处。大家在学习过程中遇到我没有表达清楚或者表述有误的地方，欢迎随时在文章下边指出，我会及时关注，随时改正。另外，大家有任何话题都可以在下边留言，我们一起交流探讨。

讨论 评论

刘畅201904211606816 1#

之前没接触过，看得一知半解

发表于 2020-01-06 22:31:27 赞(0) 回复(1)

我是祖国的花朵 N 作者 ： 加油，这块知识点确实需要有所了解才看得懂！

2020-02-17 21:17:04 赞(0) 回复(0)

请输入你的观点

回复

201810191135717 2#

大佬牛逼

发表于 2020-05-12 20:42:49 赞(0) 回复(0)

梁凉不凉 3#

之前学过兔子队列，这一块需要准备吗？还是直接看兔子好了😂

发表于 2020-05-30 15:14:47 赞(0) 回复(1)

我是祖国的花朵 作者： 有趣，都可以哈，挑选一个学习使用就可以了，简历上写一个自己学过的或者用过的。

2020-05-31 10:02:22 赞(0) 回复(0)

请输入你的观点

回复