consumer leader 选举策略 Partition leader 选举策略，为什么存在 一个broker挂了，Partition leader选举不出来

-
    -
        - 概念介绍
        - Kafka特点
        - 分区（Partition）
        - 副本
        - offset
        - 日志段
        - 代理（Broker）
        - 生产者（Producer）
        - 消费者（Consumer）
        - ISR
        - Zookeeper
        - 知识点块
        - 问答
        - Kafka相关指令
        - Zookeeper下查找命令
        - Kafka配置项
    - 遇到的坑
        - 拓展

概念介绍

- **Broker**：Kafka集群包含一个或多个服务器，这种服务器被称为broker。
- **Topic**：每条发布到Kafka集群的消息都有一个类别，这个类别被称为Topic。也可以被认为是queue
- **Partition**：分区，每个Topic包含一个或多个Partition。
- **Producer**：负责发布消息到Kafka broker
- **Consumer**：消息消费者，从Kafka broker拉取消息的客户端。
- **Consumer Group**：每一个consumer实例都属于一个consumer group，每一条消息只会被同一个consumer group里的一个consumer实例消费。（不同consumer group可以同时消费同一条消息）
- **offset**：每个partition都由一系列有序的、不可变的消息组成，这些消息被连续的追加到partition中。partition中的每个消息都有一个连续的序列号叫做offset,用于partition唯一标识一条消息
- **Replication**：副本，存在于Partition中，Leader副本负责读写，Follower副本负责从Leader拉取数据

Kafka特点

- 可水平扩展
- 高吞吐率：单机支持每秒100K条以上消息的传输
- 同时支持离线数据处理和实时数据处理
- 以时间复杂度为O(1)的方式提供消息持久化能力

分区（Partition）

每个主题的数据会被分到多个分区中，Kafka只能保证同一个分区内的数据有序性。

每个分区在物理上对应一个文件，其由 Topic-Partition 组成，分区编号从0开始。主题的分区数是可以进行配置的，方式有：

- Kafka配置文件中配置，需要重启生效
- 创建主题进行指定配置
- 客户端可以在主题创建之后，修改分区数

分区的存在使Kafka的并发能力得到了提高，理论上说，分区数越多，其吞吐量就越高。同时，分区也是实现负载均衡的基础。

副本

分区的副本的存在保证数据的可用性。从存储角度来书，分区的每个副本在逻辑上抽象为一个日志对象。分区分为Leader和Follower。为什么要存在这个的两个角色呢？若所有的分区的都是Follower，那么Kafka要与所有的Follower进行数据的读写操作，增加了开销，降低来了性能，同时，通信的对象越多，其出错的概率也就变得越高了。使用Leader和Follower机制，只需要Leader副本来对外处理客户端的读写处理，Follower副本通过同步Leader数据就可以了，这样就能保证数据的有序性和一致性。

当Leader副本失效时，会通过算法从Follower中选举出一个副本作为Leader。

需要注意哦，每个Partition的Leader副本不一定一样哦。

offset

offset是每个消息在日志文件中保存下来的一个有序的编号，新的消息会被添加到相应的Partition下的`.log`中的尾部，按序递增。

消费者消费的Offset信息被保存到了kafka的一个名为`__consumer_offsets`的主题中，其由50个Partition构成

日志段

一个日志又被划分为多个日志段（LogSegment），日志段是Kafka日志对象分片的最小单位。与日志对象一样，日志段也是一个逻辑概念，一个日志段对应磁盘上**一个具体的日志文件和两个索引人间**。日志文件以“.log”结尾，用于保存真实的消息数据，两个索引文件分别是“.index”和“.timeindex”，分别表示消息偏移量索引文件和消息时间戳索引文件。

代理（Broker）

Kafka一般是以集群的方式部署的，Broker就是Kafka Server意思。集群中的每一台服务都是Broker，你可以在同一台服务器上部署多个Broker，这个是不推荐的。每一个Broker还拥有一个brokerId，是一个非负的整数。

生产者（Producer）

负责将消息发送给Broker。

消费者（Consumer）

消费者通过Pull方式拉取数据，每个消费者都会有一个消费组（groupId），如果不指定，就默认消费组test-consumer-group，同时，每个消费者也有一个全局唯一的id，通过client.id指定，如果没有指定，根据`${groupId}-${hostName}-${timestamp}-${UUID前8位}`生成一个作为其值。一条消息在同一个消费组，只能被一个消费者消费，但可以在不同消费组中重复消费。

在实际使用中，发现使用`kafka-consumer-groups.sh`只会显示正在被当前`group.id`消费的topic信息，并不会显示所有的。

ISR

Kafka在Zookeeper中动态维护了一个ISR（In-sync Replica），即**保存同步的副本列表**，当副本存现出现宕机或者数据同步落后太多的时候，会将其从ISR中剔除。

Zookeeper

Kafka利用Zookeeper保存相应的元数据信息，如代理节点信息、Kafka集群信息、主题信息、分区信息、分区副本分配方案。动态配置信息等。帮助Kafka水平拓展及数据迁移。

知识点块

Kafka集群会保留所有的消息，无论其被消费与否。当然，因为磁盘限制，不可能永久保留所有数据（实际上也没必要），因此Kafka提供两种策略去删除旧数据。一是基于时间，二是基于partition文件大小。

因为Kafka读取特定消息的时间复杂度为O(1)，即与文件大小无关，所以这里删除文件与Kafka性能无关，选择怎样的删除策略只与磁盘以及具体的需求有关。另外，Kafka会为每一个consumer group保留一些metadata信息–当前消费的消息的position，也即offset。这个offset由consumer控制。正常情况下consumer会在消费完一条消息后线性增加这个offset。当然，consumer也可将offset设成一个较小的值，重新消费一些消息。因为offet由consumer控制，所以Kafka broker是无状态的，它不需要标记哪些消息被哪些consumer过，不需要通过broker去保证同一个consumer group只有一个consumer能消费某一条消息，因此也就不需要锁机制，这也为Kafka的高吞吐率提供了有力保障。

问答

为什么存在Partition?

> 为了使得Kafka的吞吐率可以线性提高，物理上把Topic分成一个或多个Partition，每个Partition在物理上对应一个文件夹，该文件夹下存储这个Partition的所有消息和索引文件。一个Topic可以存在于多个broker，同一个broker可以存在多个Partition，这样就可以达到Kafka的横向拓展。提高访问吞吐率，有负载均衡的效果。

topic中partition存储分布?

副本分配逻辑规则如下：

- 在Kafka集群中，每个Broker都有均等分配Partition的Leader机会。
- 上述图Broker Partition中，箭头指向为副本，以Partition-0为例:broker1中parition-0为Leader，Broker2中Partition-0为副本。
- 上述图种每个Broker(按照BrokerId有序)依次分配主Partition,下一个Broker为副本，如此循环迭代分配，多副本都遵循此规则。

副本分配算法如下：

- 将所有N Broker和待分配的i个Partition排序.
- 将第i个Partition分配到第(i mod n)个Broker上.
- 将第i个Partition的第j个副本分配到第((i + j) mod n)个Broker上.

partition中的选举?

> 为了保证数据的可靠性Kafka会给每个分区找一个节点当带头大哥（Leader），以及若干个节点当随从（Follower）。消息写入分区时，带头大哥除了自己复制一份外还会复制到多个随从。如果随从挂了，Kafka会再找一个随从，从带头大哥那里同步历史消息；如果带头大哥挂了，随从中会选举出新一任的带头大哥，继续笑傲江湖。

segment与Partition关系?

> segment对应一个文件（其实是两个，一个是数据，一个是索引），一个Partition对应一个文件，其包含了多个segment，所以可以认为是partition是对segment的封装。

消息如何选择分区?

- 如果不手动指定分区选择策略类，则会使用默认的分区策略类。
- 如果不指定消息的key，则消息发送到的分区是随着时间不停变换的。
- 如果指定了消息的key，则会根据消息的hash值和topic的分区数取模来获取分区的。
- 如果应用有消息顺序性的需要，则可以通过指定消息的key和自定义分区类来将符合某种规则的消息发送到同一个分区。同一个分区消息是有序的，同一个分区只有一个消费者就可以保证消息的顺序性消费。

如何知道需要将follower移除?

- 一个节点必须能维持与zookeeper的会话（通过zookeeper的心跳机制）
- 如果它是一个slave，它必须复制leader并且不能落后"太多"

什么时候认为消息已提交?

> 当该分区的所有同步副本已经写入到其日志中时漫该消息视为“已提交”。只有“已提交”的消息才会给到消费者。所有消费者无需担心如果leader故障，会消费到丢失的消息。因而Kafka担保在任何时候，只要至少有一个同步副本活着，承诺的消息就不会丢失。

什么时候认为消息已经被消费?

分区的follower复制机制是怎样的?

> follower可以批量的从leader复制数据，这样极大的提高复制性能（批量写磁盘），极大减少了follower与leader的差距（前文有说到，只要follower落后leader不太远，则被认为在同步列表里）。确保数据不丢失以及吞吐率

什么是复制因子?

> 复制因子就是包括了leader和follower的集合，默认是1，也就只有leader。

选举机制?

> kafka采用了一种稍微不同的方法选择quorum，而不是多数投票，kafka动态维护一组同步leader数据的副本（ISR），只有这个组的成员才有资格当选leader，kafka副本写入不被认为是已提交，直到所有的同步副本已经接收才认为。这组ISR保存在zookeeper，正因为如此，在ISR中的任何副本都有资格当选leader，这是kafka的使用模型，有多个分区和确保leader平衡是很重要的一个重要因素。有了这个模型，ISR和f+1副本，kafka的主题可以容忍f失败而不会丢失已提交的消息。如果所有的副本都死了，只能等待在ISR中的副本起死回生并选择该副本作为leader（希望它仍有所有数据）。或者选择第一个副本 （不一定在 ISR)，作为leader。

关于Controller?

> controller通过watch Zookeeper检测所有的broker failure，并负责为所有受影响的parition选举leader，再将相应的leader调整命令发送至受影响的broker。所有broker都会尝试在Zookeeper中创建/controller->{this broker id}，如果创建成功（只可能有一个创建成功），则该broker会成为controller，若创建不成功，则该broker会等待新controller的命令。

使用consumer group的好处?

> Kafka允许不同consumer group同时消费同一条消息，这一特性可以为消息的多元化处理提供了支持。实际上，Kafka的设计理念之一就是同时提供离线处理和实时处理。根据这一特性，可以使用Storm这种实时流处理系统对消息进行实时在线处理，同时使用Hadoop这种批处理系统进行离线处理，还可以同时将数据实时备份到另一个数据中心，只需要保证这三个操作所使用的consumer在不同的consumer group即可。

Consumer Rebalance?

- Kafka保证同一consumer group中只有一个consumer会消费某条消息，实际上，Kafka保证的是稳定状态下每一个consumer实例只会消费某一个或多个特定partition的数据，而某个partition的数据只会被某一个特定的consumer实例所消费。这样设计的劣势是无法让同一个consumer group里的consumer均匀消费数据，优势是每个consumer不用都跟大量的broker通信，减少通信开销，同时也降低了分配难度，实现也更简单。另外，因为同一个partition里的数据是有序的，这种设计可以保证每个partition里的数据也是有序被消费。

- 如果某consumer group中consumer数量少于partition数量，则至少有一个consumer会消费多个partition的数据，如果consumer的数量与partition数量相同，则正好一个consumer消费一个partition的数据，而如果consumer的数量多于partition的数量时，会有部分consumer无法消费该topic下任何一条消息。如果某个consumer消失了，则其管理的partition会被其他的consumer接手。

- 目前consumer rebalance的控制策略是由每一个consumer通过Zookeeper完成的。

消息Deliver guarantee?

- At most once 消息可能会丢，但绝不会重复传输
- At least one 消息绝不会丢，但可能会重复传输
- Exactly once 每条消息肯定会被传输一次且仅传输一次，很多时候这是用户所想要的。

> 接下来讨论的是消息从broker到consumer的delivery guarantee semantic。（仅针对Kafka consumer high level API）。consumer在从broker读取消息后，可以选择commit，该操作会在Zookeeper中存下该consumer在该partition下读取的消息的offset。该consumer下一次再读该partition时会从下一条开始读取。如未commit，下一次读取的开始位置会跟上一次commit之后的开始位置相同。当然可以将consumer设置为autocommit，即consumer一旦读到数据立即自动commit。如果只讨论这一读取消息的过程，那Kafka是确保了Exactly once。但实际上实际使用中consumer并非读取完数据就结束了，而是要进行进一步处理，而数据处理与commit的顺序在很大程度上决定了消息从broker和consumer的delivery guarantee semantic。

- 读完消息先commit再处理消息。这种模式下，如果consumer在commit后还没来得及处理消息就crash了，下次重新开始工作后就无法读到刚刚已提交而未处理的消息，这就对应于At most once
- 读完消息先处理再commit。这种模式下，如果处理完了消息在commit之前consumer crash了，下次重新开始工作时还会处理刚刚未commit的消息，实际上该消息已经被处理过了。这就对应于At least once。在很多情况使用场景下，消息都有一个primary key，所以消息的处理往往具有幂等性，即多次处理这一条消息跟只处理一次是等效的，那就可以认为是Exactly once。（人个感觉这种说法有些牵强，毕竟它不是Kafka本身提供的机制，而且primary key本身不保证操作的幂等性。而且实际上我们说delivery guarantee semantic是讨论被处理多少次，而非处理结果怎样，因为处理方式多种多样，我们的系统不应该把处理过程的特性–如是否幂等性，当成Kafka本身的feature）
- 如果一定要做到Exactly once，就需要协调offset和实际操作的输出。精典的做法是引入两阶段提交。如果能让offset和操作输入存在同一个地方，会更简洁和通用。这种方式可能更好，因为许多输出系统可能不支持两阶段提交。比如，consumer拿到数据后可能把数据放到HDFS，如果把最新的offset和数据本身一起写到HDFS，那就可以保证数据的输出和offset的更新要么都完成，要么都不完成，间接实现Exactly once。（目前就high level API而言，offset是存于Zookeeper中的，无法存于HDFS，而low level API的offset是由自己去维护的，可以将之存于HDFS中）

> 总之，Kafka默认保证At least once，并且允许通过设置producer异步提交来实现At most once。而Exactly once要求与目标存储系统协作，幸运的是Kafka提供的offset可以使用这种方式非常直接非常容易。

Kafka相关指令

```
# 启动kafka
bin/kafka-server-start.sh config/server.properties &
# 创建topic
bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-
factor 1 --partitions 1 --topic testTopic
# 查看topic详情
/bin/kafka-topics.sh --zookeeper=10.200.131.21:2181 --describe --topic
testTopic
# 模拟生产者
bin/kafka-console-producer.sh --broker-list localhost:9092 -- topic
testTopic
# 模拟消费者
bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic
testTopic --from-beginning
# 查看topic列表
./kafka-topics.sh --zookeeper 10.211.6.56,10.211.6.57,10.210.6.54:2181
--list
# 启动kafka
./kafka-server-start.sh -daemon /home/hadoop/app/kafka_2.11-
0.9.0.0/config/server.properties
# 删除topic
bin/kafka-topics --topic dmp.timer.output.mofang.visit --zookeeper
10.211.6.55:2181,10.211.6.56:2181,10.211.6.57:2181 --delete
# 查看partition
bin/kafka-topics.sh --zookeeper
10.211.6.55:2181,10.211.6.56:2181,10.211.6.57:2181 --describe --topic
dmp.timer.output.mofang.visit

# 主题分配副本
./kafka-topics.sh --create --zookeeper server1:2181,server2:2181 --
replication-factor 3 --partitions 6 --topic
dmp.timer.output.mofang.visit

# 查看topic partition下 offset，与group无关哦
kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list
192.168.10.23:9092 --topic dmp.gateway.formatted-dop.visit.collect --
time -1

# 查看 topic consumer消费信心，过期时间等信息
bin/kafka-console-consumer.sh --topic __consumer_offsets --bootstrap-
server localhost:9092 --formatter
"kafka.coordinator.group.GroupMetadataManager\$OffsetsMessageFormatter"
--consumer.config config/consumer.properties --from-beginning
```

Zookeeper下查找命令

```
# 获取主题分区副本分配方案
get /brokers/topics/<topic_name>

# 查看节点信息
ls /brokers/ids
# 查看leader节点
get /controller
```

Kafka配置项

```
# poll拉取限制，每次poll最大字符数
max.partition.fetch.bytes
# poll间隔时间
max.poll.interval.ms
# poll拉取限制，每次poll最大数据条数，优先级小于max.partition.fetch.bytes
max.poll.records
```

遇到的坑

- Kafka可以console中进行produce和consumer，但程序无法进行

> kafka日志满了

拓展

Exactly Once语义与事务机制原理