徐老师 • 更新于 2020-09-25

上一节 4 Kafka核心扩展... 6 Kafka生产消费... 下一节

Topic、Partition扩展

每个partition在存储层面是append log文件。
新消息都会被直接追加到log文件的尾部，每条消息在log文件中的位置称为offset(偏移量)。

越多partitions可以容纳更多的consumer,有效提升并发消费的能力。

> 具体什么时候增加topic的数量？什么时候增加partition的数量呢?

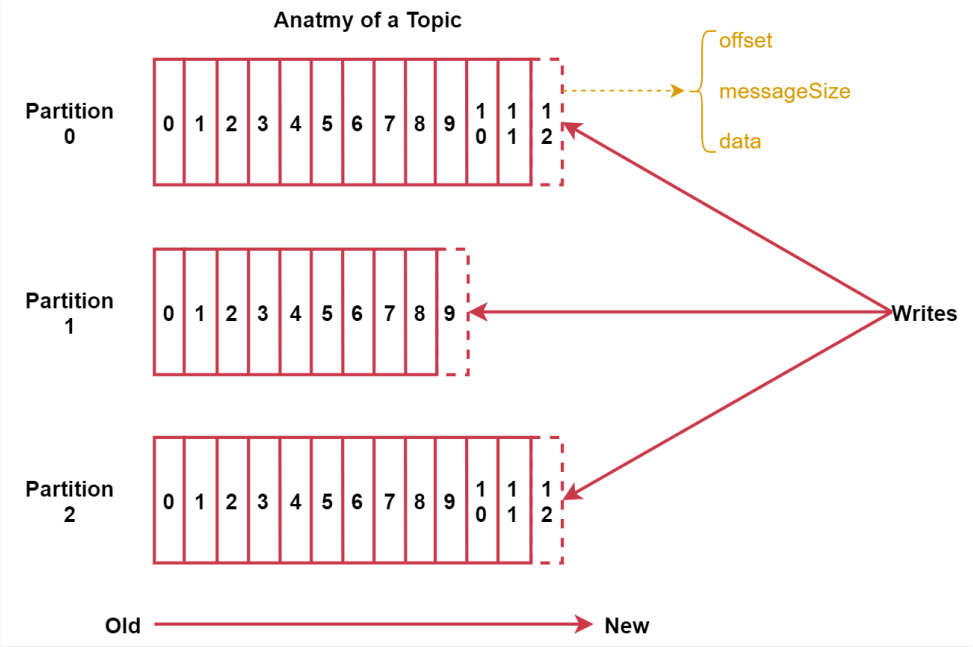
业务类型增加需要增加topic、数据量大需要增加partition

Message扩展

每条Message包含了以下三个属性：

1. offset 对应类型：long 表示此消息在一个partition中的起始的位置。可以认为offset是partition中Message的id，自增的
2. MessageSize 对应类型：int32 此消息的字节大小。
3. data 是message的具体内容。

看这个图，加深对Topic、Partition、Message的理解



存储策略

在kafka中每个topic包含1到多个partition，每个partition存储一部分Message。每条Message包含三个属性，其中有一个是offset。

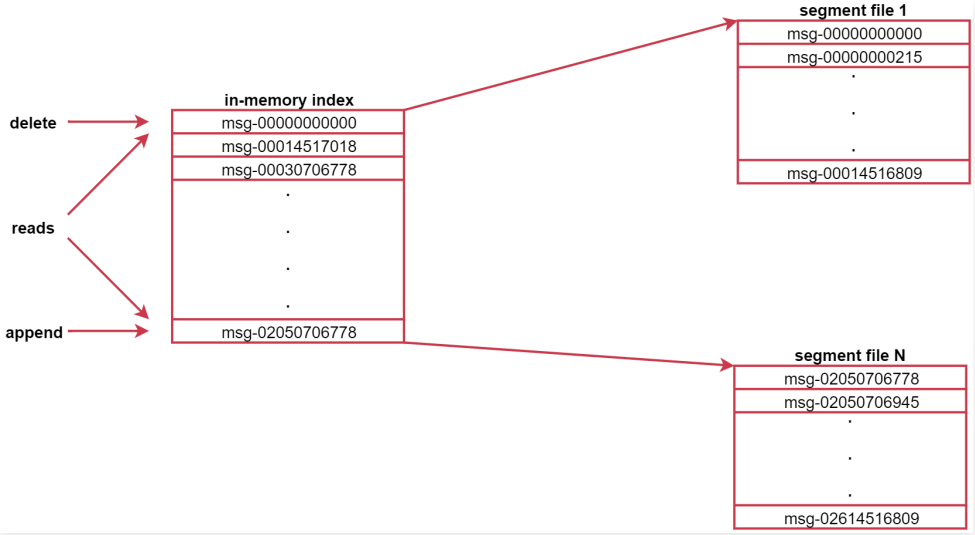
> 问题来了：offset相当于partition中这个message的唯一id，那么如何通过id高效的找到message?

两大法宝：分段+索引

kafak中数据的存储方式是这样的：
1、每个partition由多个segment【片段】组成，每个segment中存储多条消息，
2、每个partition在内存中对应一个index，记录每个segment中的第一条消息偏移量。

Kafka中数据的存储流程是这样的：
生产者生产的消息会被发送到topic的多个partition上，topic收到消息后往对应partition的最后一个segment上添加该消息，segment达到一定的大小后会创建新的segment。

来看这个图，可以认为是针对topic中某个partition的描述



图中左侧就是索引，右边是segment文件，左边的索引里面会存储每一个segment文件中第一条消息的偏移量，由于消息的偏移量都是递增的，这样后期查找起来就方便了，先到索引中判断数据在哪个segment文件中，然后就可以直接定位到具体的segment文件了，这样再找具体的那一条数据就很快了，因为都是有序的。

容错机制

> 当Kafka集群中的一个Broker节点宕机，会出现什么现象?

下面来演示一下
使用kill -9 杀掉bigdata01中的broker进程测试

代码块

```
[root@bigdata01 kafka_2.12-2.4.1]# jps
7522 Jps
2054 Kafka
1679 QuorumPeerMain
[root@bigdata01 kafka_2.12-2.4.1]# kill 2054
```

我们可以先通过zookeeper来查看一下，因为当kafka集群中的broker节点启动之后，会自动向zookeeper中进行注册，保存当前节点信息

代码块

```
.....
[zk: localhost:2181(CONNECTED) 1] ls /brokers
[ids, seqid, topics]
[zk: localhost:2181(CONNECTED) 2] ls /brokers/ids
[1, 2]
```

此时发现zookeeper的/brokers/ids下面只有2个节点信息
可以通过get命令查看节点信息，这里面会显示对应的主机名和端口号

代码块

```
[zk: localhost:2181(CONNECTED) 4] get /brokers/ids/1
{"listener_security_protocol_map":{"PLAINTEXT":"PLAINTEXT"},"endpoints":["PLA
```

然后再使用describe查询topic的详细信息，会发现此时的分区的leader全部变成了目前存活的另外两个节点
此时可以发现Isr中的内容和Replicas中的不一样了，因为Isr中显示的是目前正常运行的节点
所以当Kafka集群中的一个Broker节点宕机之后，对整个集群而言没有什么特别的大影响，此时集群会给partition重新选出来一些新的Leader节点

代码块

```
[root@bigdata01 kafka_2.12-2.4.1]# bin/kafka-topics.sh --describe --zookeeper
Topic: hello    PartitionCount: 5       ReplicationFactor: 2    Configs:
        Topic: hello    Partition: 0    Leader: 2       Replicas: 2,0       I
        Topic: hello    Partition: 1    Leader: 1       Replicas: 0,1       I
        Topic: hello    Partition: 2    Leader: 1       Replicas: 1,2       I
        Topic: hello    Partition: 3    Leader: 2       Replicas: 2,1       I
        Topic: hello    Partition: 4    Leader: 2       Replicas: 0,2       I
```

> 当Kafka集群中新增一个Broker节点，会出现什么现象?

新加入一个broker节点，zookeeper会自动识别并在适当的机会选择此节点提供服务
再次启动bigdata01节点中的broker进程测试

代码块

```
[root@bigdata01 kafka_2.12-2.4.1]# bin/kafka-server-start.sh -daemon config/s
```

此时到zookeeper中查看一下

代码块

```
[root@bigdata01 apache-zookeeper-3.5.8-bin]# bin/zkCli.sh
Connecting to localhost:2181
.....
[zk: localhost:2181(CONNECTED) 1] ls /brokers
[ids, seqid, topics]
[zk: localhost:2181(CONNECTED) 2] ls /brokers/ids
[0, 1, 2]
```

发现broker.id为0的这个节点信息也有了

在通过describe查看topic的描述信息，Isr中的信息和Replicas中的内容是一样的了

代码块

```
[root@bigdata01 kafka_2.12-2.4.1]# bin/kafka-topics.sh --describe --zookeeper
igs:
0       I
```

```
4           Topic: hello    Partition: 1    Leader: 1       Replicas: 0,1       I
5           Topic: hello    Partition: 2    Leader: 1       Replicas: 1,2       I
6           Topic: hello    Partition: 3    Leader: 2       Replicas: 2,1       I
7           Topic: hello    Partition: 4    Leader: 2       Replicas: 0,2       I
```

> 但是启动后有个问题：发现新启动的这个节点不会是任何分区的leader？怎么重新均匀分配呢？

1、Broker中的自动均衡策略（默认已经有）

代码块

```
1    auto.leader.rebalance.enable=true
2    leader.imbalance.check.interval.seconds 默认值：300
```

2、手动执行：

代码块

```
1    bin/kafka-leader-election.sh --bootstrap-server localhost:9092 --election-typ
```

我们尝试使用手工执行

代码块

```
1    [root@bigdata01 kafka_2.12-2.4.1]# bin/kafka-leader-election.sh --bootstrap-s
2    Successfully completed leader election (PREFERRED) for partitions hello-4, he
```

执行后的效果如下，这样就实现了均匀分配

代码块

```
1    [root@bigdata01 kafka_2.12-2.4.1]# bin/kafka-topics.sh --describe --zookeeper
2    Topic: hello    PartitionCount: 5       ReplicationFactor: 2    Configs:
3           Topic: hello    Partition: 0    Leader: 2       Replicas: 2,0Isr: 2,0
4           Topic: hello    Partition: 1    Leader: 0       Replicas: 0,1Isr: 1,0
5           Topic: hello    Partition: 2    Leader: 1       Replicas: 1,2Isr: 1,2
6           Topic: hello    Partition: 3    Leader: 2       Replicas: 2,1Isr: 2,1
7           Topic: hello    Partition: 4    Leader: 0       Replicas: 0,2Isr: 2,0
```

4 Kafka核心扩展内容 上一节 下一节 6 Kafka生产消费者实战

我要提出意见反馈

意见反馈 收藏教程 标记书签