优先副本的选举

分区使用多副本机制来提升可靠性，但只有 leader 副本对外提供读写服务，而 follower 副本只负责在内部进行消息的同步。如果一个分区的 leader 副本不可用，那么就意味着整个分区变得不可用，此时就需要 Kafka 从剩余的 follower 副本中挑选一个新的 leader 副本来继续对外提供服务。虽然不够严谨，但从某种程度上说，broker 节点中 leader 副本个数的多少决定了这个节点负载的高低。

在创建主题的时候，该主题的分区及副本会尽可能均匀地分布到 Kafka 集群的各个 broker 节点上，对应的 leader 副本的分配也比较均匀。比如我们使用 kafka-topics.sh 脚本创建一个分区数为3、副本因子为3的主题 topic-partitions，创建之后的分布信息如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
describe --topic topic-partitions
Topic:topic-partitions  PartitionCount:3
ReplicationFactor:3 Configs:
    Topic: topic-partitions Partition: 0
Leader: 1   Replicas: 1,2,0 Isr: 1,2,0
    Topic: topic-partitions Partition: 1
Leader: 2   Replicas: 2,0,1 Isr: 2,0,1
    Topic: topic-partitions Partition: 2
Leader: 0   Replicas: 0,1,2 Isr: 0,1,2
```

可以看到 leader 副本均匀分布在 brokerId 为0、1、2的 broker 节点之中。针对同一个分区而言，同一个 broker 节点中不可能出现它的多个副本，即 Kafka 集群的一个 broker 中最多只能有它的一

个副本，我们可以将 leader 副本所在的 broker 节点叫作分区的 leader 节点，而 follower 副本所在的 broker 节点叫作分区的 follower 节点。

随着时间的更替，Kafka 集群的 broker 节点不可避免地会遇到宕机或崩溃的问题，当分区的 leader 节点发生故障时，其中一个 follower 节点就会成为新的 leader 节点，这样就会导致集群的负载不均衡，从而影响整体的健壮性和稳定性。当原来的 leader 节点恢复之后重新加入集群时，它只能成为一个新的 follower 节点而不再对外提供服务。比如我们将 brokerId 为2的节点重启，那么主题 topic-partitions 新的分布信息如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
describe --topic topic-partitions
Topic:topic-partitions  PartitionCount:3
ReplicationFactor:3 Configs:
    Topic: topic-partitions Partition: 0
Leader: 1   Replicas: 1,2,0 Isr: 1,0,2
    Topic: topic-partitions Partition: 1
Leader: 0   Replicas: 2,0,1 Isr: 0,1,2
    Topic: topic-partitions Partition: 2
Leader: 0   Replicas: 0,1,2 Isr: 0,1,2
```

可以看到原本分区1的 leader 节点为2，现在变成了0，如此一来原本均衡的负载变成了失衡：节点0的负载最高，而节点2的负载最低。

为了能够有效地治理负载失衡的情况，Kafka 引入了优先副本（preferred replica）的概念。所谓的优先副本是指在AR集合列表中的第一个副本。比如上面主题 topic-partitions 中分区0的AR集合列表（Replicas）为[1,2,0]，那么分区0的优先副本即为1。理想情况下，优先副本就是该分区的leader 副本，所以也可以称之为

preferred leader。Kafka 要确保所有主题的优先副本在 Kafka 集群中均匀分布，这样就保证了所有分区的 leader 均衡分布。如果 leader 分布过于集中，就会造成集群负载不均衡。

所谓的优先副本的选举是指通过一定的方式促使优先副本选举为 leader 副本，以此来促进集群的负载均衡，这一行为也可以称为“分区平衡”。

需要注意的是，分区平衡并不意味着 Kafka 集群的负载均衡，因为还要考虑集群中的分区分配是否均衡。更进一步，每个分区的 leader 副本的负载也是各不相同的，有些 leader 副本的负载很高，比如需要承载 TPS 为30000的负荷，而有些 leader 副本只需承载个位数的负荷。也就是说，就算集群中的分区分配均衡、leader 分配均衡，也并不能确保整个集群的负载就是均衡的，还需要其他一些硬性的指标来做进一步的衡量，这个会在后面的章节中涉及，本节只探讨优先副本的选举。

在 Kafka 中可以提供分区自动平衡的功能，与此对应的 broker 端参数是 auto.leader. rebalance.enable，此参数的默认值为 true，即默认情况下此功能是开启的。如果开启分区自动平衡的功能，则 Kafka 的控制器会启动一个定时任务，这个定时任务会轮询所有的 broker 节点，计算每个 broker 节点的分区不平衡率（broker 中的不平衡率=非优先副本的 leader 个数/分区总数）是否超过 leader.imbalance.per.broker.percentage 参数配置的比值，默认值为10%，如果超过设定的比值则会自动执行优先副本的选举动作以求分区平衡。执行周期由参数 leader.imbalance.check.interval.seconds 控制，默认值为300秒，即5分钟。

不过在生产环境中不建议将 auto.leader.rebalance.enable 设置为默认的 true，因为这可能引起负面的性能问题，也有可能引起客户端一定时间的阻塞。因为执行的时间无法自主掌控，如果在关键时期（比如电商大促波峰期）执行关键任务的关卡上执行优先副本的自动选举操作，势必会有业务阻塞、频繁超时之类的风险。前面也分析

过，分区及副本的均衡也不能完全确保集群整体的均衡，并且集群中一定程度上的不均衡也是可以忍受的，为防止出现关键时期“掉链子”的行为，笔者建议还是将掌控权把控在自己的手中，可以针对此类相关的埋点指标设置相应的告警，在合适的时机执行合适的操作，而这个“合适的操作”就是指手动执行分区平衡。

Kafka 中 kafka-perferred-replica-election.sh 脚本提供了对分区 leader 副本进行重新平衡的功能。优先副本的选举过程是一个安全的过程，Kafka 客户端可以自动感知分区 leader 副本的变更。下面的示例演示了 kafka-perferred-replica-election.sh 脚本的具体用法：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
preferred-replica-election.sh --zookeeper
localhost:2181/kafka

Created preferred replica election path with
topic-demo-3,__consumer_offsets-22, topic-config-
1,__consumer_offsets-30,__bigdata_monitor-
12,__consumer_offsets-8,__consumer_offsets-
21,topic-create-0,__consumer_offsets-4,topic-
demo-1,topic-partitions-1,__consumer_offsets-
27,__consumer_offsets-7,__consumer_offsets-
9,__consumer_offsets-46,(…省略若干)

[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
describe --topic topic-partitions
Topic:topic-partitions  PartitionCount:3
ReplicationFactor:3 Configs:
    Topic: topic-partitions Partition: 0
Leader: 1   Replicas: 1,2,0 Isr: 1,0,2
    Topic: topic-partitions Partition: 1
Leader: 2   Replicas: 2,0,1 Isr: 0,1,2
    Topic: topic-partitions Partition: 2
Leader: 0   Replicas: 0,1,2 Isr: 0,1,2
```

可以看到在脚本执行之后，主题 topic-partitions 中的所有 leader 副本的分布已经和刚创建时的一样了，所有的优先副本都成为 leader 副本。

上面示例中的这种使用方式会将集群上所有的分区都执行一遍优先副本的选举操作，分区数越多打印出来的信息也就越多。leader 副本的转移也是一项高成本的工作，如果要执行的分区数很多，那么必然会对客户端造成一定的影响。如果集群中包含大量的分区，那么上面

的这种使用方式有可能会失效。在优先副本的选举过程中，具体的元数据信息会被存入 ZooKeeper 的/admin/preferred_replica_election 节点，如果这些数据超过了 ZooKeeper 节点所允许的大小，那么选举就会失败。默认情况下 ZooKeeper 所允许的节点数据大小为1MB。

kafka-perferred-replica-election.sh 脚本中还提供了 path-to-json-file 参数来小批量地对部分分区执行优先副本的选举操作。通过 path-to-json-file 参数来指定一个 JSON 文件，这个 JSON 文件里保存需要执行优先副本选举的分区清单。

举个例子，我们再将集群中 brokerId 为2的节点重启，不过我们现在只想对主题 topic- partitions 执行优先副本的选举操作，那么先创建一个JSON文件，文件名假定为 election.json，文件的内容如下：

```
{
        "partitions":[
                {
                        "partition":0,
                        "topic":"topic-
partitions"
                },
                {
                        "partition":1,
                        "topic":"topic-
partitions"
                },
                {
                        "partition":2,
                        "topic":"topic-
partitions"
                }
        ]
}
```

然后通过 kafka-perferred-replica-election.sh 脚本配合 path-to-json-file 参数来对主题 topic-partitions 执行优先副本的选举操作，具体示例如下：

```
[root@node1 kafka_2.11-2.0.0]# bin/kafka-
preferred-replica-election.sh --zookeeper
localhost:2181/kafka --path-to-json-file
election.json
Created preferred replica election path with
topic-partitions-0,topic-partitions-1, topic-
partitions-2
Successfully started preferred replica election
for partitions Set(topic- partitions-0, topic-
partitions-1, topic-partitions-2)

[root@node1 kafka_2.11-2.0.0]# bin/kafka-
topics.sh --zookeeper localhost:2181/ kafka --
describe --topic topic-partitions
Topic:topic-partitions  PartitionCount:3
ReplicationFactor:3 Configs:
    Topic: topic-partitions Partition: 0
Leader: 1   Replicas: 1,2,0 Isr: 1,0,2
    Topic: topic-partitions Partition: 1
Leader: 2   Replicas: 2,0,1 Isr: 0,1,2
    Topic: topic-partitions Partition: 2
Leader: 0   Replicas: 0,1,2 Isr: 0,1,2
```

读者可以自行查看一下集群中的其他主题是否像之前没有使用 path-to-json-file 参数的一样也被执行了选举操作。

在实际生产环境中，一般使用 path-to-json-file 参数来分批、手动地执行优先副本的选举操作。尤其是在应对大规模的 Kafka 集群时，理应杜绝采用非 path-to-json-file 参数的选举操作方式。同时，优先副本的选举操作也要注意避开业务高峰期，以免带来性能方面的负面影响。