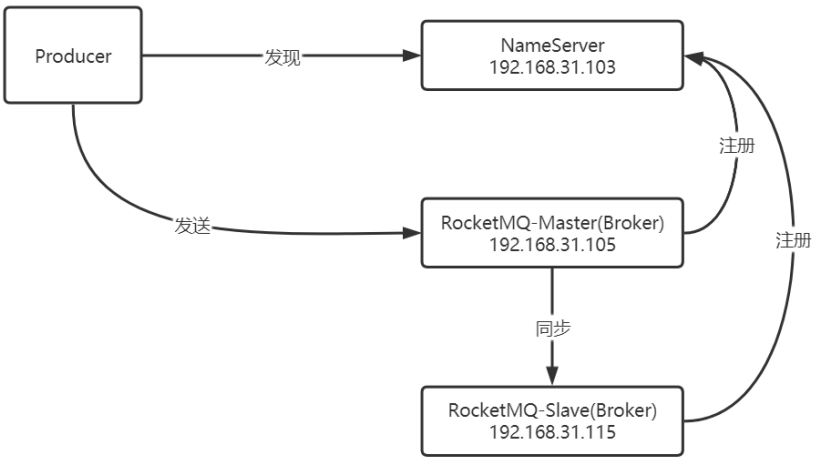
集群消费模式与广播消费模式

环境准备



生产者CmProducer

生产者是一致的，循环生成10条普通消息投给给Broker，主题为：cm-sample-data ，Tag：test ，Key：n

```
@Slf4j
public class CmProducer {
    public static void main(String[] args) {
        //DefaultMQProducer用于发送非事务消息
        DefaultMQProducer producer = new DefaultMQProducer("cm-producer-group");
        //注册NameServer地址
        producer.setNamesrvAddr("192.168.31.103:9876");
        //异步发送失败后Producer自动重试2次
        producer.setRetryTimesWhenSendAsyncFailed(2);
        try {
            //启动生产者实例
            producer.start();
            for(Integer i = 0 ; i < 10 ; i++) {
                //消息数据
                String data = "第" + i + "条消息数据";
                //消息主题
                Message message = new Message("cm-sample-data", "test", i.toString(), data.getBytes());
                //发送结果
                SendResult result = producer.send(message);
                log.info("Broker响应： " + result);
            }
        }catch (Exception e){
            e.printStackTrace();
        }finally {
            try {
```

```
            //关闭连接
            producer.shutdown();
            log.info("连接已关闭");
        }catch (Exception e){
            e.printStackTrace();
        }
    }
  }
}
```

集群模式消费者

代码分析

```
@Slf4j
public class CmClusterConsumer {

    public static void main(String[] args) throws Exception {

        // 声明并初始化一个 consumer
        // 需要一个 consumer group 名字作为构造方法的参数
        DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("cm-
cluster-consumer-group");

        // 同样也要设置 NameServer 地址，须要与提供者的地址列表保持一致
        consumer.setNamesrvAddr("192.168.31.103:9876");

        //设置为集群模式（负载均衡）
        consumer.setMessageModel(MessageModel.CLUSTERING);

        // 设置 consumer 所订阅的 Topic 和 Tag，*代表全部的 Tag
        consumer.subscribe("cm-sample-data", "*");

        // 注册消息监听者
        consumer.registerMessageListener(new MessageListenerConcurrently() {
            @Override
            public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> list,
ConsumeConcurrentlyContext consumeConcurrentlyContext) {
                list.forEach(msg->{
                    log.info("收到消息：" + new String(msg.getBody()));
                });

                // 返回消费状态
                // CONSUME_SUCCESS 消费成功
                // RECONSUME_LATER 消费失败，需要稍后重新消费
                return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
            }
        });
```

```
        // 调用 start() 方法启动 consumer
        consumer.start();
        log.info("集群消费者启动成功，正在监听新消息");
    }
}
```

运行结果

启动1-4个实例：

实例1：

```
21:54:58.944 [main] INFO com.itlaoqi.rocketmq.consumemode.CmClusterConsumer - 集群消费者启动成功，正在监听新消息
21:55:08.963 [ConsumeMessageThread_3] INFO com.itlaoqi.rocketmq.consumemode.CmClusterConsumer - 收到消息：第2条消息数据
21:55:08.979 [ConsumeMessageThread_5] INFO com.itlaoqi.rocketmq.consumemode.CmClusterConsumer - 收到消息：第6条消息数据
```

实例2：

```
21:55:01.010 [main] INFO com.itlaoqi.rocketmq.consumemode.CmClusterConsumer - 集群消费者启动成功，正在监听新消息
21:55:08.949 [ConsumeMessageThread_1] INFO com.itlaoqi.rocketmq.consumemode.CmClusterConsumer - 收到消息：第0条消息数据
21:55:08.949 [ConsumeMessageThread_3] INFO com.itlaoqi.rocketmq.consumemode.CmClusterConsumer - 收到消息：第4条消息数据
21:55:08.985 [ConsumeMessageThread_4] INFO com.itlaoqi.rocketmq.consumemode.CmClusterConsumer - 收到消息：第8条消息数据
```

实例3：

```
21:55:02.987 [main] INFO com.itlaoqi.rocketmq.consumemode.CmClusterConsumer - 集群消费者启动成功，正在监听新消息
21:55:08.965 [ConsumeMessageThread_1] INFO com.itlaoqi.rocketmq.consumemode.CmClusterConsumer - 收到消息：第1条消息数据
21:55:08.978 [ConsumeMessageThread_2] INFO com.itlaoqi.rocketmq.consumemode.CmClusterConsumer - 收到消息：第5条消息数据
21:55:08.988 [ConsumeMessageThread_3] INFO com.itlaoqi.rocketmq.consumemode.CmClusterConsumer - 收到消息：第9条消息数据
```

实例4：

```
21:55:04.490 [main] INFO com.itlaoqi.rocketmq.consumemode.CmClusterConsumer - 集群消费者启动成功，正在监听新消息
21:55:08.978 [ConsumeMessageThread_1] INFO com.itlaoqi.rocketmq.consumemode.CmClusterConsumer - 收到消息：第3条消息数据
21:55:08.982 [ConsumeMessageThread_2] INFO com.itlaoqi.rocketmq.consumemode.CmClusterConsumer - 收到消息：第7条消息数据
```

广播模式消费者

源码分析

只有setMessageModel方法传入BROADCASTING常量，其他没有任何变化

```
@Slf4j
public class CmBroadcastConsumer {

    public static void main(String[] args) throws Exception {
        //...其余代码完全一样
        //设置为广播模式
        consumer.setMessageModel(MessageModel.BROADCASTING);
        //...其余代码完全一样
    }
}
```

运行结果

1-4个实例均消费到10条消息，不过不同实例之间获取消息的前后顺序均有差别。

```
21:59:10.398 [main] INFO
com.itlaoqi.rocketmq.consumemode.CmBroadcastConsumer - 广播消费者启动成功，正在监听新消息
21:59:16.379 [ConsumeMessageThread_5] INFO
com.itlaoqi.rocketmq.consumemode.CmBroadcastConsumer - 收到消息：第4条消息数据
21:59:16.380 [ConsumeMessageThread_3] INFO
com.itlaoqi.rocketmq.consumemode.CmBroadcastConsumer - 收到消息：第2条消息数据
21:59:16.380 [ConsumeMessageThread_1] INFO
com.itlaoqi.rocketmq.consumemode.CmBroadcastConsumer - 收到消息：第0条消息数据
21:59:16.380 [ConsumeMessageThread_4] INFO
com.itlaoqi.rocketmq.consumemode.CmBroadcastConsumer - 收到消息：第3条消息数据
21:59:16.380 [ConsumeMessageThread_2] INFO
com.itlaoqi.rocketmq.consumemode.CmBroadcastConsumer - 收到消息：第1条消息数据
21:59:16.381 [ConsumeMessageThread_6] INFO
com.itlaoqi.rocketmq.consumemode.CmBroadcastConsumer - 收到消息：第5条消息数据
21:59:16.386 [ConsumeMessageThread_7] INFO
com.itlaoqi.rocketmq.consumemode.CmBroadcastConsumer - 收到消息：第6条消息数据
21:59:16.389 [ConsumeMessageThread_8] INFO
com.itlaoqi.rocketmq.consumemode.CmBroadcastConsumer - 收到消息：第7条消息数据
21:59:16.399 [ConsumeMessageThread_9] INFO
com.itlaoqi.rocketmq.consumemode.CmBroadcastConsumer - 收到消息：第8条消息数据
21:59:16.399 [ConsumeMessageThread_10] INFO
com.itlaoqi.rocketmq.consumemode.CmBroadcastConsumer - 收到消息：第9条消息数据
```

Process finished with exit code -1