MQ四兄弟：如何保证消息可靠性

RabbitMQ消息可靠机制

在RabbitMQ中，消息的可靠性传输可以通过以下几种机制来保证：

1. **持久化**：确保消息在RabbitMQ重启后不会丢失。
2. **确认机制**：确确保消息从生产者到RabbitMQ以及从RabbitMQ到消费者都被成功处理。
3. **重试机制**：处理消息消费失败后的重试和死信队列（DLX）。

以下是一些关键机制和相应的Java代码片段：

1. 持久化（Durability）

持久化包括将队列和消息设置为持久化，这样在RabbitMQ重启后消息仍然存在。

```
// 声明持久化队列
boolean durable = true;
channel.queueDeclare(QUEUE_NAME, durable, false, false, null);
String message = "Hello, RabbitMQ!";
// 发送持久化消息
channel.basicPublish("", QUEUE_NAME,
MessageProperties.PERSISTENT_TEXT_PLAIN, message.getBytes("UTF-8"));
```

2. 确认机制（Acknowledgements）

RabbitMQ提供了手动ACK机制。生产者确认（Publisher Confirms）和消费者确认（Consumer Acknowledgements）机制。

生产者确认

生产者可以开启确认模式，确保消息已被RabbitMQ接收到并持久化。

```
channel.queueDeclare(QUEUE_NAME, true, false, false, null);
channel.confirmSelect();//启用了生产者确认模式
String message = "Hello, RabbitMQ!";
channel.basicPublish("", QUEUE_NAME,
MessageProperties.PERSISTENT_TEXT_PLAIN, message.getBytes("UTF-8"));
```

消费者确认

消费者可以在处理完消息后发送确认信号，以确保消息被成功处理。

```
// 处理消息
try {
    doWork(message);
} finally {
System.out.println(" [x] Done");
    // 消费者手动确认消息
    channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);
}
};
boolean autoAck = false; // 自动确认关闭
channel.basicConsume(QUEUE_NAME, autoAck, deliverCallback,
consumerTag -> { });
```

3. 重试机制和死信队列（DLX）

RabbitMQ支持死信队列（Dead Letter Exchange），可以将处理失败的消息重新路由到另一个死信队列。

```
try (Connection connection = factory.newConnection(); Channel channel = connection.createChannel()) {
    // 声明死信队列
    channel.queueDeclare(DLX_NAME, true, false, false, null);

    Map<String, Object> args = new HashMap<>();
    args.put("x-dead-letter-exchange", "");
    // 设置队列的参数，使其使用死信交换器
    args.put("x-dead-letter-routing-key", DLX_NAME);

    // 声明正常队列，并绑定死信队列
    channel.queueDeclare(QUEUE_NAME, true, false, false, args);

    String message = "Hello, RabbitMQ!";
    channel.basicPublish("", QUEUE_NAME, MessageProperties.PERSISTENT_TEXT_PLAIN, message.getBytes("UTF-8"));
}
}
}
```

消费者可以通过channel.basicReject或channel.basicNack拒绝消息，并设置requeue参数为false，使消息进入死信队列。

```
// 拒绝消息并不重新入队
channel.basicReject(deliveryTag, false);
```

RocketMQ消息可靠机制

在RocketMQ中，消息的可靠性传输可以通过以下几种机制来保证：

1. **消息持久化**：确保消息在Broker重启后不会丢失。
2. **同步刷盘**：确保消息持久化到磁盘后才确认。
3. **消息确认机制**：确保消息被正确处理。
4. **重试机制和死信队列**：处理消息发送失败后的重试和死信队列（DLX）。

以下是一些关键机制和相应的Java代码片段：

1. 消息持久化

RocketMQ默认情况下会持久化消息。 RocketMQ 不需要和rabbitmq一样显式的设置，也就是说，当您发送消息到 RocketMQ 时，消息会自动持久化到磁盘，而无需显式地设置消息为持久化。

2. 消息确认机制

RocketMQ确保消息被消费者正确处理，消费者需要对消息进行确认。

生产者

```
 // 发送消息并同步等待结果
                SendResult sendResult = producer.send(msg);
                // 确认消息是否成功发送
                if (sendResult.getSendStatus() == SendStatus.SEND_OK) {
                    System.out.printf("Message sent successfully: %s%n",
sendResult);
                } else {
                    System.out.printf("Message sending failed: %s%n",
sendResult);
                }
```

消费者

```
consumer.registerMessageListener(new MessageListenerConcurrently() {
    @Override
    public ConsumeConcurrentlyStatus
consumeMessage(List<MessageExt> msgs,
ConsumeConcurrentlyContext context) {
        for (MessageExt msg : msgs) {
            System.out.printf("%s Receive New Messages: %s %n",
Thread.currentThread().getName(), new String(msg.getBody()));
            // 处理消息
        }
        // 返回消息消费状态
        return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
    }
});
```

3. 消息重试机制

RocketMQ支持消息发送失败后的重试机制，生产者可以配置发送重试次数。

```
// 设置消息发送失败后的重试次数
producer.setRetryTimesWhenSendFailed(3);
// 启动生产者实例
producer.start();
```

4. 配置同步刷盘模式

在RocketMQ的配置文件中，可以将同步刷盘模式设置为`SYNC_MASTER`，以确保消息被同步到磁盘后才确认。

```
brokerRole = SYNC_MASTER
flushDiskType = SYNC_FLUSH
```

Kafka消息可靠机制

在Kafka中，消息的可靠性传输通过以下几种机制来保证：

1. **消息持久化**：消息被写入磁盘，以防止数据丢失。
2. **复制机制**：在Kafka集群中，消息被复制到多个Broker上。
3. **确认机制**：确保消息被正确写入。
4. **重试机制**：处理消息发送失败后的重试。

以下是一些关键机制和相应的Java代码片段：

1. 消息持久化和确认机制

Kafka的生产者可以配置`acks`参数来确保消息被可靠地写入。常用的`acks`配置包括：

* `acks=0`：生产者不会等待任何Broker的确认。
* `acks=1`：生产者会等待主节点（Leader）的确认。
* `acks=all`：生产者会等待所有副本节点的确认，确保消息被复制到所有副本。

生产者示例

```
    // 设置生产者配置
    Properties props = new Properties();
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
    props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
    props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
    // 设置acks为all，以确保消息被所有副本确认
    props.put(ProducerConfig.ACKS_CONFIG, "all");
    // 设置重试次数
    props.put(ProducerConfig.RETRIES_CONFIG, 3);
    // 设置每次重试的间隔时间
```

```
    props.put(ProducerConfig.RETRY_BACKOFF_MS_CONFIG, 100);

    Producer<String, String> producer = new KafkaProducer<>(props);

    try {
        for (int i = 0; i < 10; i++) {
            String message = "Hello Kafka " + i;
            ProducerRecord<String, String> record = new
ProducerRecord<>("my_topic", Integer.toString(i), message);
            // 发送消息并等待结果
            Future<RecordMetadata> future = producer.send(record);
            RecordMetadata metadata = future.get();
        }
}
```

消费者示例：

```
// 关闭自动提交偏移量
    props.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG,
"false");
    // 设置自动提交偏移量的间隔时间
    props.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG,
"1000");

    KafkaConsumer<String, String> consumer = new
KafkaConsumer<>(props);
    consumer.subscribe(Collections.singletonList("my_topic"));

    try {
        while (true) {
            ConsumerRecords<String, String> records =
consumer.poll(Duration.ofMillis(100));
            for (ConsumerRecord<String, String> record : records) {
                System.out.printf("Consumed message with offset: %d,
partition: %d%n", record.offset(), record.partition());
                // 手动提交偏移量
                consumer.commitSync();
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        consumer.close();
```

```
    }
```

2. 配置消息的重试机制

Kafka生产者通过配置重试机制来处理消息发送失败的情况：

```
props.put(ProducerConfig.RETRIES_CONFIG, 3);
props.put(ProducerConfig.RETRY_BACKOFF_MS_CONFIG, 100);
```

Pulsar消息可靠机制

Apache Pulsar通过以下几种机制来保证消息的可靠性传输：

1. **消息持久化**：消息被持久化到磁盘，以防止数据丢失。
2. **复制机制**：在Pulsar集群中，消息被复制到多个Broker上。
3. **确认机制**：确保消息被消费者正确处理。
4. **重试机制**：处理消息发送失败后的重试。

以下是一些关键机制和相应的Java代码片段：

1. 消息持久化和复制机制

Pulsar默认情况下会持久化消息并复制到多个Broker上，以确保高可靠性。以下是一个生产者的示例，展示了如何创建一个Pulsar生产者并发送消息。

```
    // 创建生产者，并设置生产者属性
    Producer<byte[]> producer = client.newProducer(Schema.BYTES)
            .topic("my-topic")
            .sendTimeout(0, TimeUnit.SECONDS) // 发送无超时限制
            .producerName("my-producer")
            .enableBatching(false) // 禁用批处理，以减少延迟
```

```
            .create();

    // 发送消息并同步等待结果
    String message = "Hello, Pulsar!";
    producer.send(message.getBytes());

    System.out.println("Message sent successfully");

    // 关闭生产者和客户端
    producer.close();
    client.close();
}
}
```

2. 消费**确认机制**

Pulsar消费者确保消息被正确处理并确认。以下是一个消费者的示例，展示了如何创建一个Pulsar消费者并处理消息。

```
    // 创建消费者，并设置消费者属性
    Consumer<byte[]> consumer = client.newConsumer(Schema.BYTES)
            .topic("my-topic")
            .subscriptionName("my-subscription")
            .subscriptionType(SubscriptionType.Exclusive)
            .subscribe();

    // 消费消息并确认
    while (true) {
        Message<byte[]> msg = consumer.receive();
        try {
            System.out.printf("Message received: %s%n", new
String(msg.getData()));
            // 手动确认消息
            consumer.acknowledge(msg);
        } catch (Exception e) {
            // 处理消息失败，负确认消息
            consumer.negativeAcknowledge(msg);
        }
    }
}
```

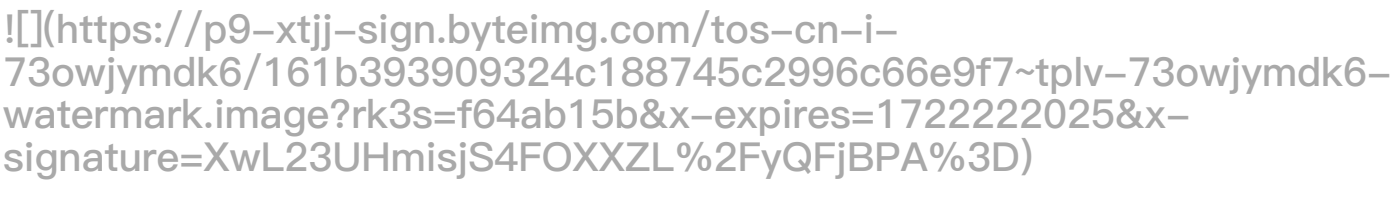
3. 配置重试机制

Pulsar的生产者可以配置重试机制来处理消息发送失败的情况。

```
  // 创建生产者实例
        Producer<byte[]> producer = client.newProducer()
                .topic("my-topic") // 设置主题
                .sendTimeout(10, java.util.concurrent.TimeUnit.SECONDS) //
设置发送超时时间
                .maxPendingMessages(1000) // 设置最大挂起消息数
                .messageRoutingMode(MessageRoutingMode.SinglePartition)
// 设置消息路由模式
                .enableBatching(true) // 启用批处理
                .batchingMaxPublishDelay(1,
java.util.concurrent.TimeUnit.MILLISECONDS) // 批处理最大发布延迟
                .blockIfQueueFull(true) // 当消息队列满时阻塞
                .retryLetterTopic("my-topic-retry") // 设置重试主题
                .deadLetterTopic("my-topic-dlq") // 设置死信队列主题
                .enableRetry(true) // 启用重试机制
                .initialSequenceId(0) // 设置初始序列ID
                .create();

```

总结
--



可以发现消息中间件如 RabbitMQ、RocketMQ、Kafka 和 Pulsar 在保证消息可靠性方面都差不多，都是从消息的生产者、MQ本身、消费者三个方向来保证的。

* 消息持久化机制
* 消息确认机制
* 重试机制
* 死信队列