# 83_性能优化：基于 MQ 对完成订单模块进行异步化改造，提升系统性能



## 1、开篇

上节课延续上节课创建订单的后续流程，完成了对应的代码实现。主要介绍了 OrderController、OrderServiceImpl、OrderMapper 中对应代码的修改，其中包括支付订单、完成订单、评论订单的业务代码。本节课会思考上述代码会有哪些问题，并提出基于 MQ 对完成订单模块异步化改造的方案，从而提升系统性能。今天课程的内容包括以下几个部分：

- 同步处理订单后续流程存在的问题
- 如何通过异步方式提高系统性能

## 2、同步处理订单后续流程存在的问题

上节课介绍了订单创建以后会经历支付订单、完成订单和评论订单几个过程。通常来说在支付订单以后教师会上门教学教学完成以后会完成订单，和取消订单的逻辑相似，这里也应该有主动完成订单：也就是点击“完成订单”按钮；超时完成订单：超过教学时间 N 天后（N 可以根据业务规则进行配置）。对于超出限期的订单系统会自动监控，并且对其执行完成操作。

如图 1 所示，和超时取消订单如出一辙的是，支付订单以后需要建立一个 Job 通过规则定义不断检测订单的状态和支付时间与当前时间的关系。要判断订单状态为已支付，同时还要判断支付时间是否小于当前时间减去等待时间，这个等待时间是可以自行配置的。和延迟取消订单不同的是，这里的等待时间会更加长一些，可能需要按照天为单位记录，不过我们在进行代码讲解的时候为了方便测试会将时间缩短，这个会在后面的代码演示中做说明。

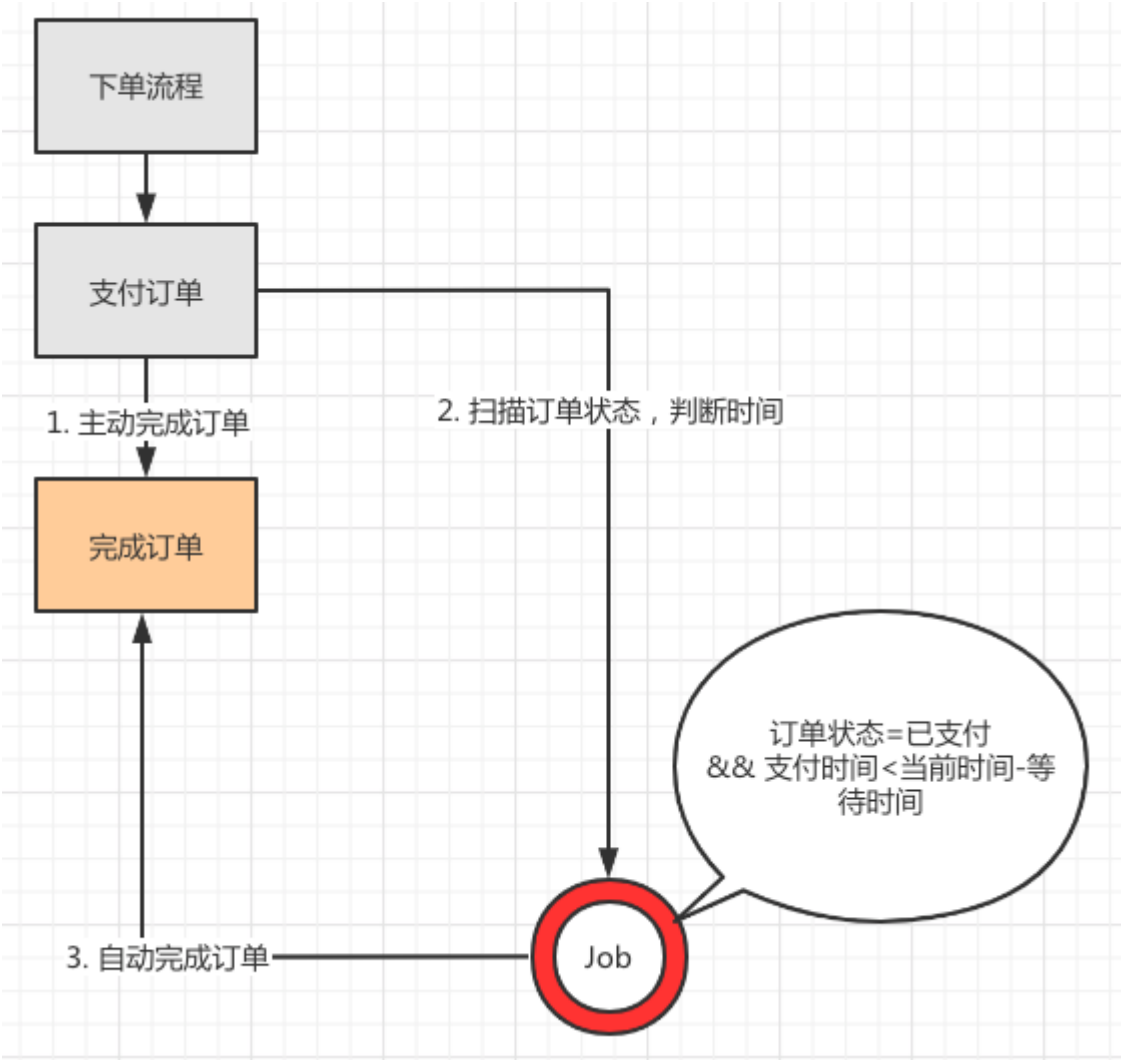


图 1 定时 Job 的方式扫描订单的状态

很明显如果使用扫描订单状态的方式会消耗大量的系统性能，并不是最佳的实现方案，于是就有了下面的异步方式。

### 3、如何通过异步方式提高系统性能

说起异步的处理方式，不由得想起上周介绍的取消订单，它的处理场景与支付订单很相似也存在处理延迟消息的情况，于是我们如法炮制给支付订单到完成订单也设计了类似的方案。

如图 2 所示，下单流程完毕以后会进入支付订单的环节，这里分为两个部分超时处理和主动处理，我们来一起看看：

1. 支付订单完成以后会发送延迟消息，该消息发送到 RocketMQ 队列中。
2. 延迟消息消费者，在一段时间以后消费队列中的消息。这个时间可能是教学完成的几天以后，也或者更久。
3. 在判断消息类型是完成延迟订单以后，会进行完成订单的操作，包括：发优惠券和增加积分。
4. 如果在支付订单以后，点击“完成订单“按钮也会发送一个消息到 RocketMQ，这是一个即时消息，之所以在这里设计为消息的机制是为了让支付订单和完成订单两个功能进行解耦。
5. 消息消费者会消费队列中的完成订单消息，这里判断消息的类型就与上面不同了，这里判断的是完成订单的消息类型而不是延迟完成订单的消息类型。
6. 当满足条件了以后也会进行完成订单的操作。

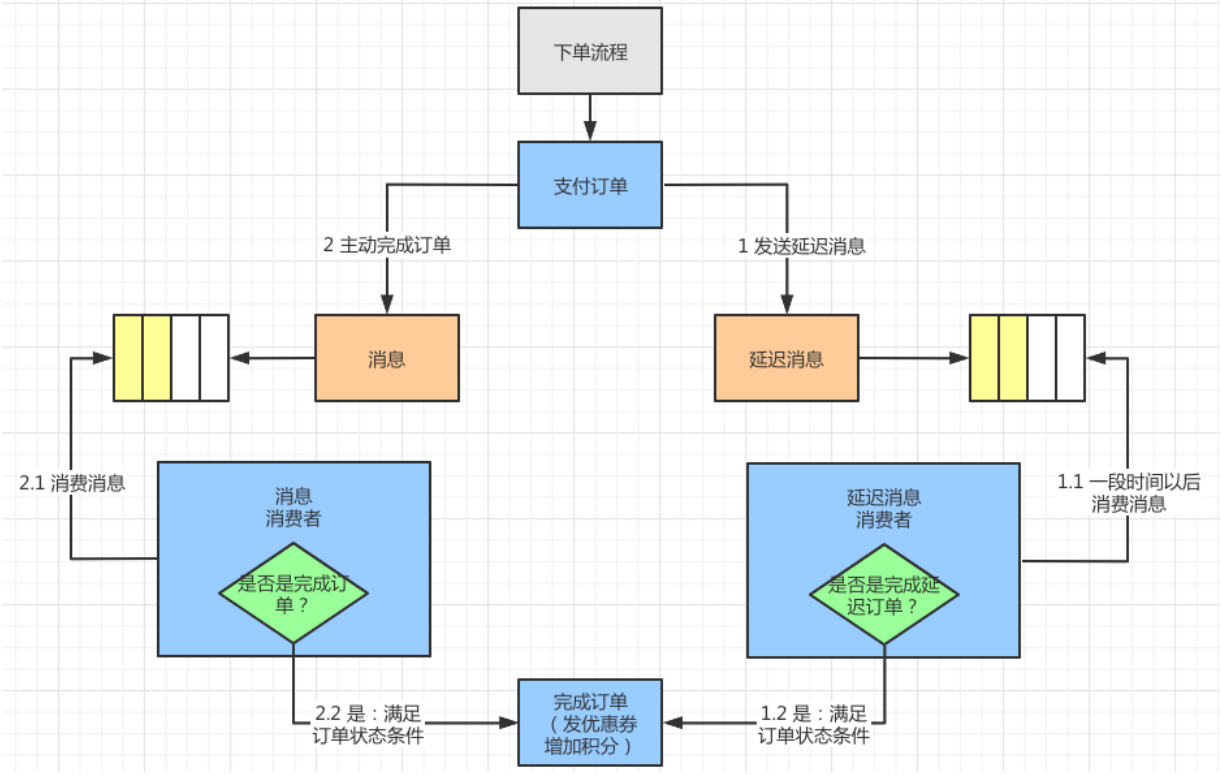


图 2 异步处理支付订单的流程图

## 4、总结

本节课首先讲述了支付订单到完成订单的业务流程，为了完成这一过程需要引入 Job 不断检查订单状态和支付的相关时间参数，这样会带来系统性能的问题。又通过取消订单也遇到类似的场景，联想到使用 RocketMQ 进行异步操作，从而提高性能也解耦了支付订单和完成订单的功能，于是引出了队列处理支付订单的方案。该方案中有两条执行流程，分别使用了延迟消息和及时消息来完成。

下节课会针对本节课将方案落地，带大家进行编码实现。下期见，拜拜。