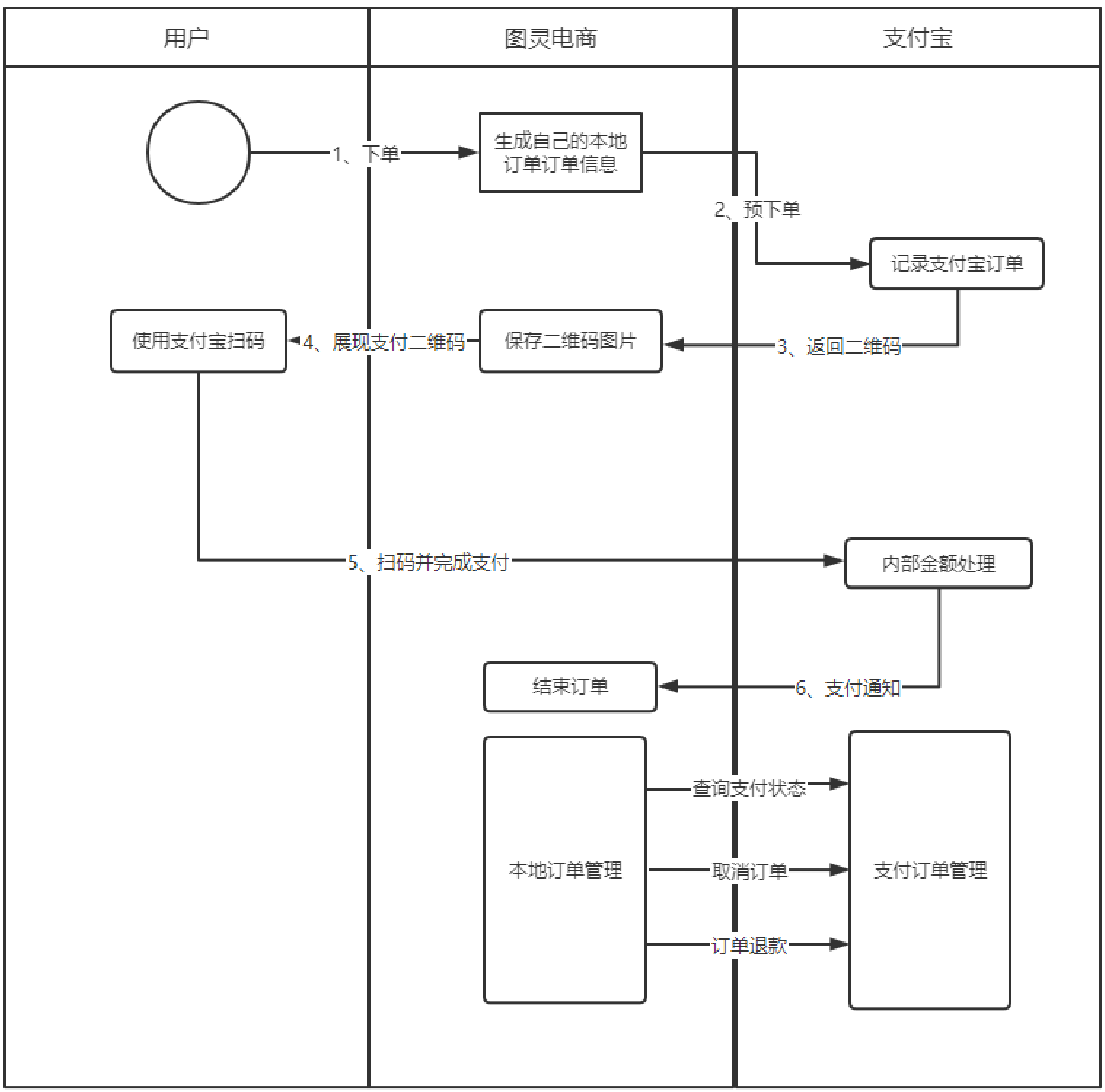
使用RocketMQ优化订单超时取消流程

电商项目优化订单支付流程

1、梳理支付宝当面付的预下单流程

在熟悉了支付宝的接入方式之后，再来思考在电商场景下，要如何接入支付宝来提供订单支付功能。结合图灵电商的业务场景，自然能够想到这样的接入流程：



1、用户在图灵电商上选好商品，从购物车开始选择下单。

2、图灵电商记录用户的订单数据后，会往支付宝进行一次预下单，可以理解为申请一个支付订单。

3、如果预下单没有问题的话，支付宝会给图灵电商同步返回一个二维码图片。

4、图灵电商将二维码图片保存到本地服务器上，并与自己的业务订单建立绑定关系。

5、图灵电商将二维码图片展现给用户，用户使用自己的支付宝扫描二维码，完成登录、支付的一系列操作。

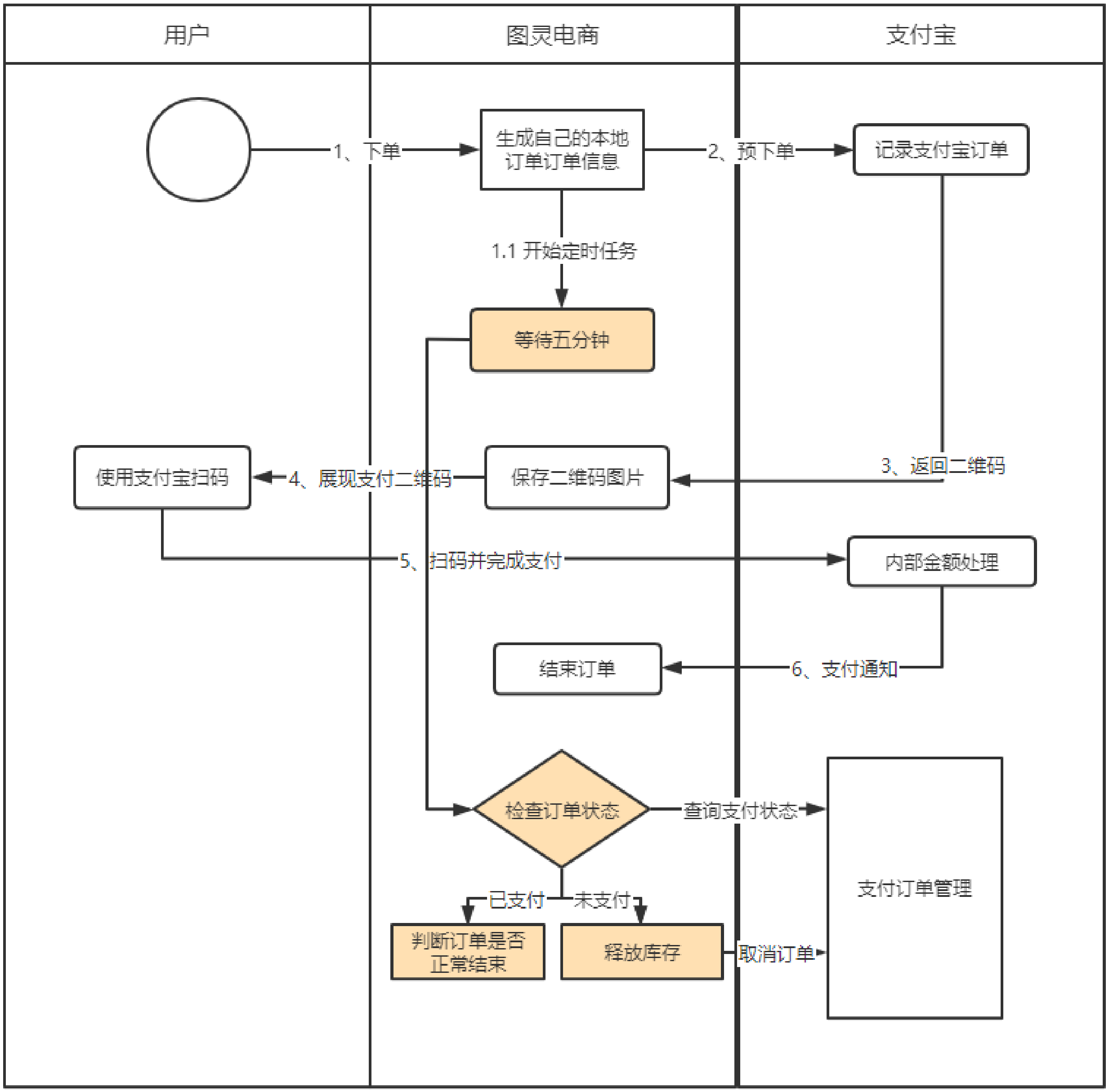
6、支付完成后，支付宝会给图灵电商发送一个异步通知，告知订单支付完成。图灵电商接收到这个通知，就可以完成订单后续的业务操作。

主流程是没有什么问题，基本上就照搬刚才的流程就可以了。但是，就按照这个流程实现订单管理，你会发现有个小问题，对于流程图下方的订单取消操作还没有进行设计。而实际上，图灵电商项目跟网上其他的电商项目都是一样，用户下完订单后是需要及时进行支付的。电商项目在用户下单时就需要锁定商品库存，如果用户长期不支付，锁定的商品就无法正常销售。

所以，通常对于订单都会设定一个支付时间，比如五分钟内需要完成支付。如果没有支付，就需要取消订单，释放库存。那应该如何设计订单的超时判断流程？

2、使用延迟任务实现支付超时判断

有朋友说，那简单，做个定时任务，五分钟后去支付宝查一下订单是否完成了支付。



1、下单时增加一个定时任务，在五分钟后对订单进行超时判断。

2、超时判断时，可以先去支付宝上查询订单支付状态。

如果已支付，则判断订单是否正常结束，这是因为在用户完成扫码支付后，支付宝正常会往图灵电商发送支付成功的通知。但是这个通知是没有事务保证的，所以是非常有可能失败的，这时就需要在订单超时判断时对状态进行对齐。

如果未支付，则需要释放库存，取消本地订单，然后通知支付宝取消支付订单。

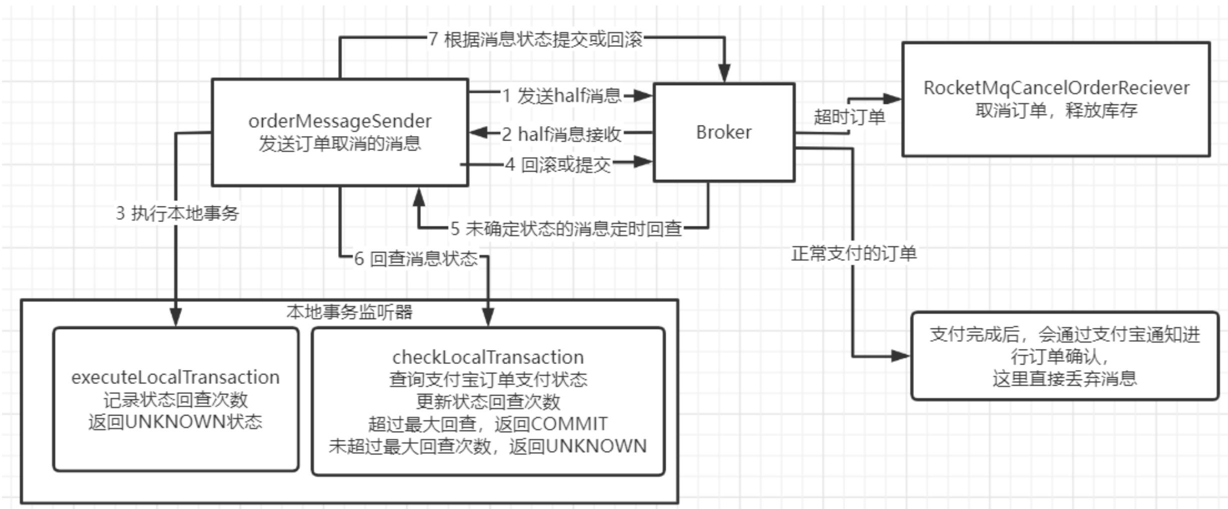
这种设计方式很自然。但是会有一个小问题，就是对订单状态的判断会不及时。订单支付状态只有在五分钟后的超时判断时才能最终确定，这就不太及时。这样对于一些并发量比较高的场景就不太合适。比如在12306抢火车票，是不是你一支付完成，12306马上就知道了？就给你发火车票了？或者你回顾一下日常使用支付宝进行支付的场景，是不是你一支付，商家马上就知道了？更别说对于秒杀等超高并发的场景了。

那要如何优化呢？通常企业中的做法并不会等到订单超时时才去查询订单状态，而是在后台会多次频繁查询支付宝支付状态，这样可以更及时的获得支付结果。例如五分钟超时时间，至少需要半分钟或者一分钟去查一次支付宝订单状态，如果支付成功了，就及时结束后续等待处理过程。如果没有完成支付，就再开启下一个定时任务，等待下次检查。

这样一分析，你会不会觉得这个定时任务光是流程控制就有点麻烦了？再跟业务逻辑绑定在一起，这个任务的逻辑会变得非常复杂。有没有现成的框架可以帮我们来优化这个复杂的定时任务逻辑，让我们只要专心关注业务逻辑呢？有，RocketMQ的事务消息就是一个比较好的工具。

3、用RocketMQ事务消息改造支付超时判断流程。

RocketMQ的事务消息机制在VIP课程中做过详细讲解，很多同学表示流程大概懂，但是不知道具体怎么用。RocketMQ事务消息机制的核心是对消息状态进行不断的确认。循环确认的过程就正好可以用来改造，解决上面说的频繁任务调度的问题。这样就可以专注于开发业务逻辑，而不用关注频繁复杂的任务调度逻辑。



在图灵电商项目中，是在向支付宝进行预下单时，发送一条事务消息。只不过这里发送的消息，是用来通知下游服务进行本地订单取消的。

下面结合项目代码进行流程分解

核心代码流程：

1、支付宝预下单时发送事务消息。 OmsPortalOrderController#tradeQrCode： 使用 orderMessageSender.sendCreateOrderMsg(orderId,memberId); 发送消息，这个消息实际上是用来通知下游服务进行订单取消的。

2、发送消息后，就会先执行本地事务。 TransactionListenerImpl#executeLocalTransaction方法。在这个方法中会将订单ID放到Redis中，这样可以在后续进行支付状态检查时，快速找到对应的业务信息。只要下单成功，就会返回UNKOWN状态，这样RocketMQ会在之后进行状态回查。

3、然后在事务状态回查时，会执行 TransactionListenerImpl#checkLocalTransaction方法。在这个方法里会自行记录回查次数，超过最大次数就直接取消订单。`注意，这里最大回查次数需要根据业务要求进行定制。`

如果没有超过最大次数，就可以去支付宝中查询订单支付状态。

如果已经支付完成，则返回ROLLBACK状态，消息取消，后续就不会再进行本地订单取消了。

如果未支付，则记录回查次数后，返回UNKNOWN状态，等待下次回查。

4、如果事务消息最终发送出去，也就是订单已经超时，就会将消息发送到RocketMQ的 ${rocketmq.tulingmall.asyncOrderTopic}这个Topic下。下游的消费者RocketMqCancelOrderReciever就会完成取消本地订单，释放库存等操作。

> 补充一个小知识点，RocketMQ的事务消息回查间隔可以通过参数 transactionCheckInterval 定制

在这个流程中，表面上利用RocketMQ的事务消息机制将频繁的定时任务拆解成事务回查的过程，实际上是通过不断的事务回查来确保分布式事务的最终一致性。

流程扩展：

当前图灵电商的实现流程，实际上就是一个基础，在面临更多更复杂的业务场景时，还需要对业务层面的细节问题进行详细设计。例如：

1、通过聚合支付进行分布式事务控制：当前图灵电商项目，只完成了与支付宝的对接，而在对接过程中，是直接使用支付宝的二维码通知用户进行当面支付。而用户使用支付宝扫码支付的过程，图灵电商都是完全不知道的，也就没有办法对用户的支付动作进行控制。比如如果图灵电商本地的订单已经超时，就要阻止用户进行扫码支付。当前项目的处理方式是在支付宝的回调接口判断订单状态，如果订单式已关闭，则发起订单回退。这样显然效率是不高的。

在很多电商项目中，会采用聚合支付的方式，统一对接多个第三方支付方。用户的支付动作就不是直接与支付宝这样的第三方支付公司交互完成，而是要经过电商后台转发请求完成。这时，就可以通过添加一些分布式锁机制，保证整个支付业务是串行执行的，以防止在电商进行订单超时回退后，用户再次扫码支付。

2、正向通知与反向通知： 当前图灵电商项目中，是通过事务消息通知下游服务订单取消，这其实就是一种反向通知的方式。但是其实最直观的方式还是使用正向通知，即通过事务消息通知下游服务进行订单支付确认，这样这个下单的消息就容易扩展更多的下游消费者。结合图灵电商，订单下单确认是用户完成支付后，支付宝发起的通知来确认的。这时，如果订单确认的下游服务实现了幂等控制，就完全可以将事务消息机制改为正向通知。即在事务消息回查过程中，确认用户已经完成了支付，就发送消息通知下游服务订单支付成功。这样也可以防止支付宝通知丢失造成的订单状态缺失。

而用户订单超时判断，则可以在事务消息的checkLocalTransaction状态回查过程中，通过记录回查次数判断。如果已经超时，则返回Rollback。同时启动另外一个消息生产者，往下游服务发送一个订单取消的消息，这样也是可以的。

3、兜底补偿机制： 例如在当前图灵电商项目中，对于订单超时后的回退处理，不光通过RocketMQ的事务消息进行了通知，另外也部署了一个定时任务，批量回退超时的订单。

```java
/**
 * 订单超时取消并解锁库存的定时器
 */
@Component
public class OrderTimeOutCancelTask {
    private Logger LOGGER =LoggerFactory.getLogger(OrderTimeOutCancelTask.class);
    @Autowired
    private OmsPortalOrderService portalOrderService;

    /**
     * cron表达式：Seconds Minutes Hours DayofMonth Month DayofWeek [Year]
     * 每10分钟扫描一次，扫描设定超时时间之前下的订单，如果没支付则取消该订单
     */
    @Scheduled(cron = "0 0/10 * ? * ?")
    private void cancelTimeOutOrder(){
        CommonResult result = portalOrderService.cancelTimeOutOrder();
        LOGGER.info("取消订单，并根据sku编号释放锁定库存:{}",result);
    }
```

```
}
```

在这个任务中，会以十分钟为间隔，对超过超时时间未支付的订单进行统一的撤回操作。这其实就是一种事务消息的兜底补偿机制，以处理那些事务消息机制有可能漏处理的超时订单。在设计金融相关业务时，这种兜底策略会显得尤为重要。

> 有道云笔记链接： https://note.youdao.com/s/ChShTWK7