## 整体构件

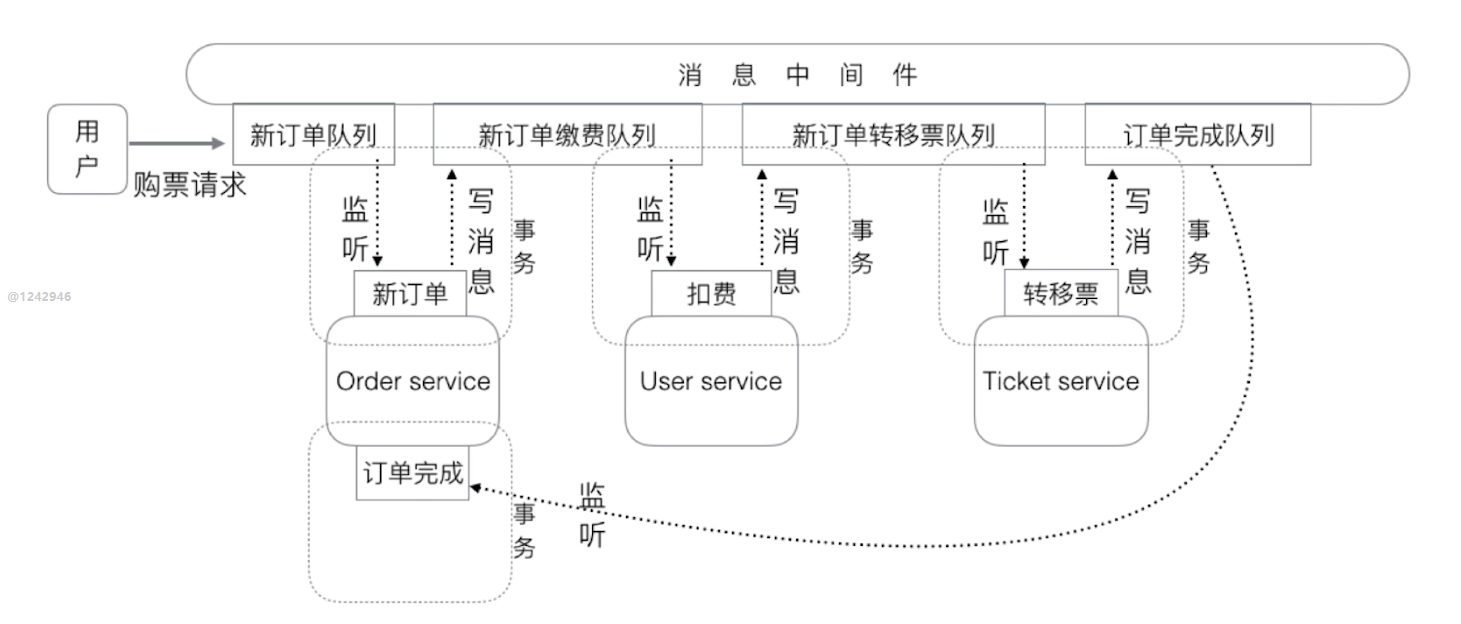
1, User service, Order Service, Ticket Service

2. ActiveMQ作为消息中间件

3. 错误处理，定时任务检查超时并回滚

4. 幂等性,实现方法的幂等性

## 业务流程图



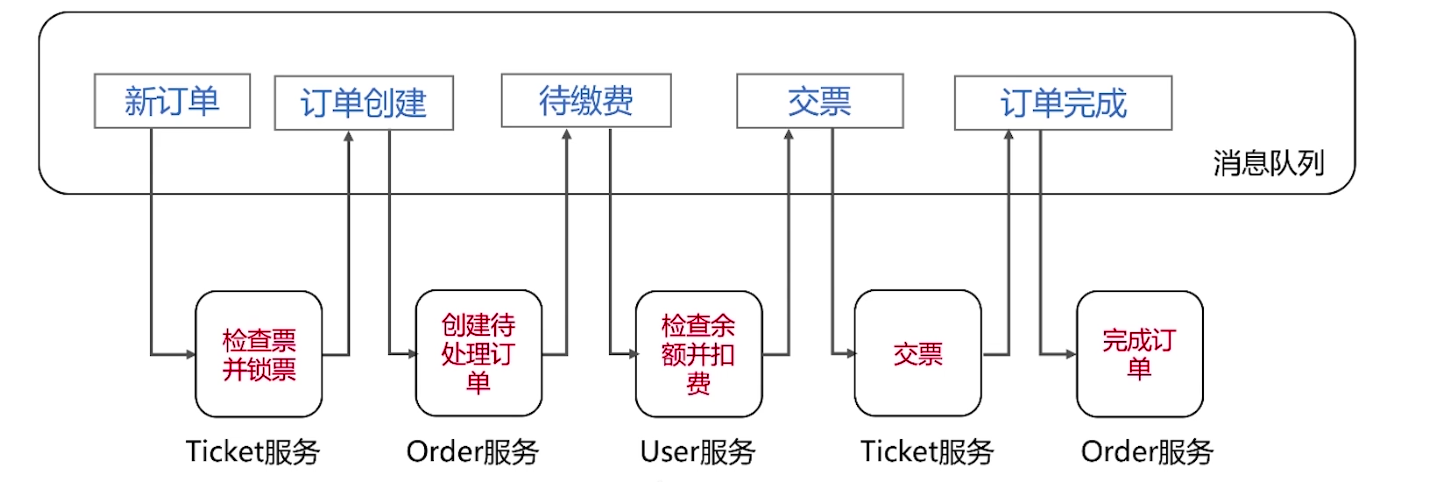
由消息驱动将各个服务的操作串起来。

每一个服务都是从MQ读消息，做业务操作然后写消息。每一个服务都是依赖Spring的事务同步机制把JMS的事务和数据库的事务同步到一起，尽量实现两个数据源分布式事务的同步性。由于不能完全的保证它的事务性，就有可能数据库事务提交，MQ的写消息事务没有提交就回滚，就有可能触发读消息重发的情况，所以在每个服务里都要实现幂等性，以处理读消息重试的情况。

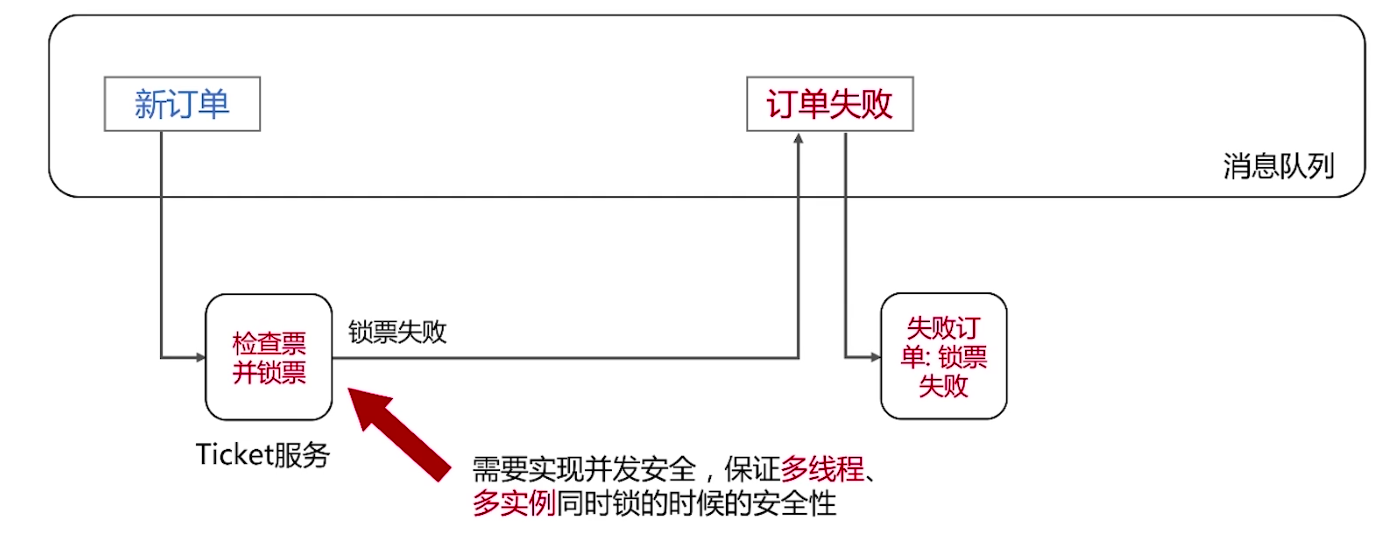
## 购票业务流程

1. 检查购买的业务条件：票，余额等
2. 创建订单
3. 扣费
4. 交票
5. 完成订单

## 消息流程



## 消息流程-锁票失败

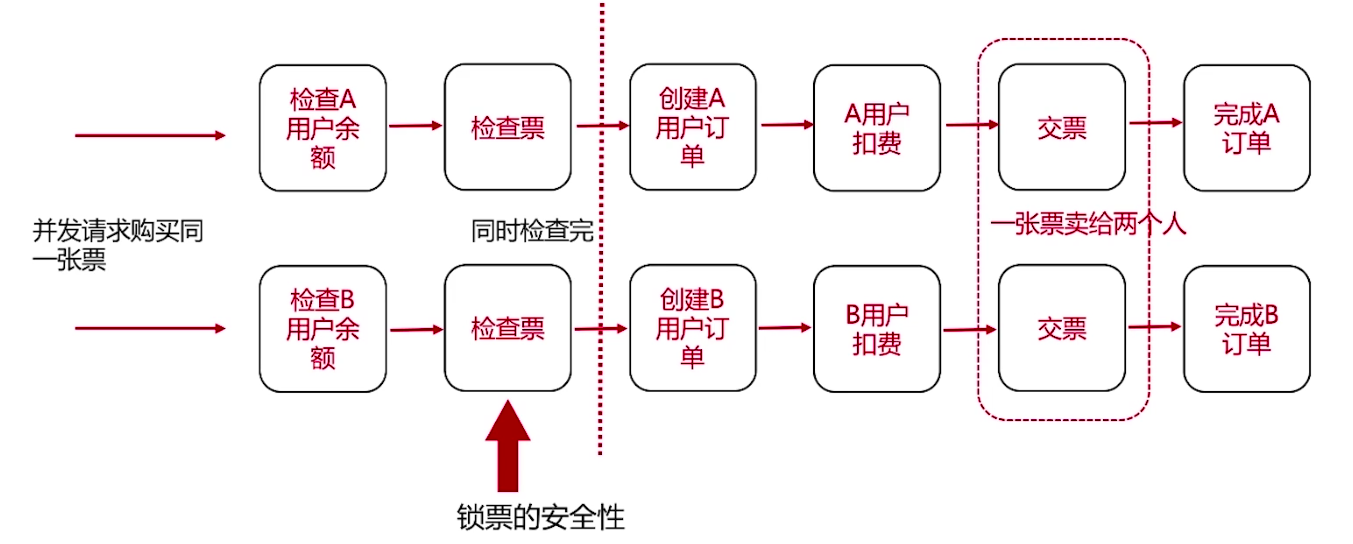


在消息驱动模式下，锁票失败将发送订单失败的消息至“订单失败”的队列里，然后Order服务去响应去记一个失败的订单，锁票失败这么一个状态。

P.S.: 一般情况下，锁票失败就当场返给用户，锁票失败，请重新选择另外的座位等操作，但我们的设计是消息驱动的，新订单的创建是写到MQ里的，这个消息的响应异步的去处理，把结果写到另外一个地方，用户是不能马上拿到票的状态的结果的，所以，锁票失败的时候写了一个失败的订单，然后用户就可以在他订单的结果里面看到订单的状态。

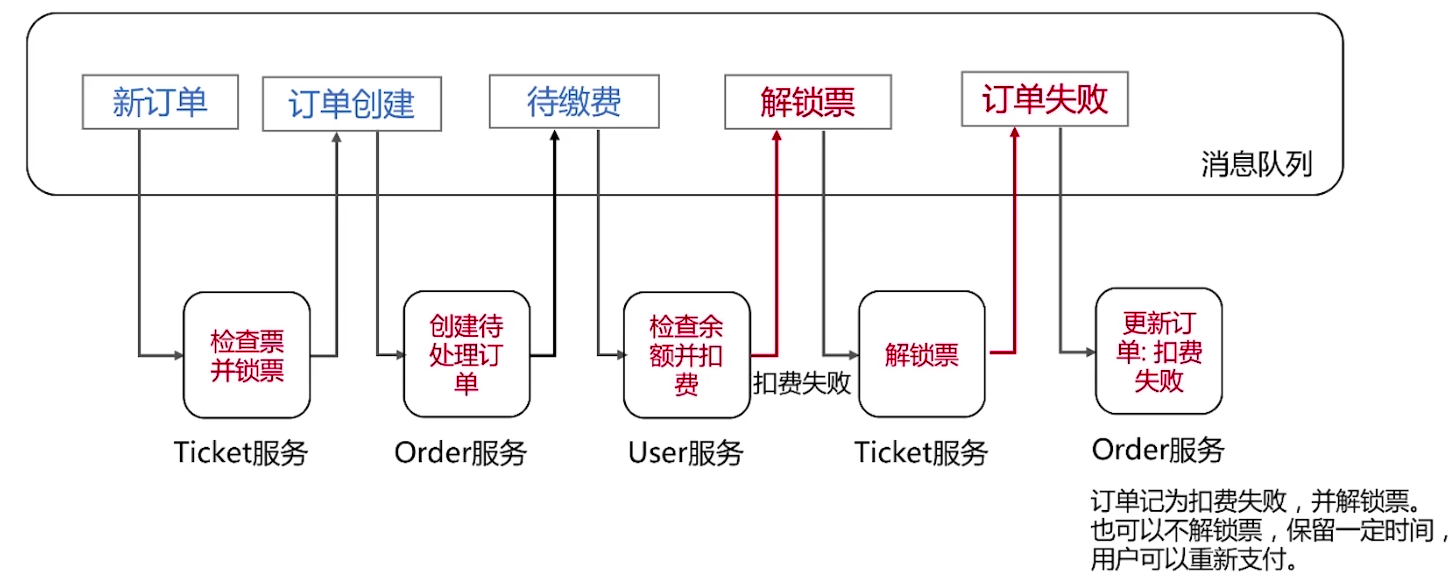
如果开始的时候用户的订单请求不是写到MQ里而是直接交由Ticket服务去处理，然后在Ticket服务里检查票啊等操作，然后如果说锁票失败了，那么就可以直接的返回给用户，这样其实也可以，但是呢对于消息驱动的事务实现来说，这是一种模式；我们尽量不要把一般的直接调用服务的模式与这种模式混在一起，这样很容易出现一些其他的问题。

## 购票业务流程



分布式多实例情况下的分布式安全性。

## 消息流程-扣费失败



## 异常订单处理

1. 定时处理异常订单
2. 未被处理完成: 某一个服务所有实例宕机，请求就有可能一直不能被处理，需要定时任务做一些回滚操作。
3. 未被按失败订单处理
4. 解锁票，撤销交票
5. 对于余额之类的重要数据，可能使用人工处理。

## 锁票操作在分布式系统下的安全性

1. 设置@JmsListener（concurrency=”1”）设置一个消费者，来做单线程的处理；故不适用于多实例。

某些MQ支持制定一些规则，满足这些规则的消息由特定的consumer来处理。

1. 使用事务或数据库的锁的一些特性。

在事务当中数据是被锁住的，只有事务提交后数据才会被释放，这种情况下其他的修改就不会成功。锁在不同的DB里实现可能不一样，H2只有表级锁，Mysql支持行级锁。

1. 分布式锁

在开始某一个Service方法之前，用一个分布式的锁去锁住它，我们可以在某一个票的ID上去锁它，所以要买这一张票的请求就能实现分布式的安全。分布式锁需要注意数据库的二级缓存。

**SpringCloud+Redis实现分布式锁：**

<https://blog.csdn.net/forezp/article/details/68957681>