Saga是分布式事务领域最有名气的解决方案之一，最初出现在1987年Hector Garcaa-Molrna & Kenneth Salem发表的论文SAGAS里。

Saga是由一系列的本地事务构成。每一个本地事务在更新完数据库之后，会发布一条消息或者一个事件来触发Saga中的下一个本地事务的执行。如果一个本地事务因为某些业务规则无法满足而失败，Saga会执行在这个失败的事务之前成功提交的所有事务的补偿操作。

Saga的实现有很多种方式，其中最流行的两种方式是：

- 基于事件的方式。这种方式没有协调中心，整个模式的工作方式就像舞蹈一样，各个舞蹈演员按照预先编排的动作和走位各自表演，最终形成一只舞蹈。处于当前Saga下的各个服务，会产生某类事件，或者监听其它服务产生的事件并决定是否需要针对监听到的事件做出响应。
- 基于命令的方式。这种方式的工作形式就像一只乐队，由一个指挥家（协调中心）来协调大家的工作。协调中心来告诉Saga的参与方应该执行哪一个本地事务。

基于事件的方式

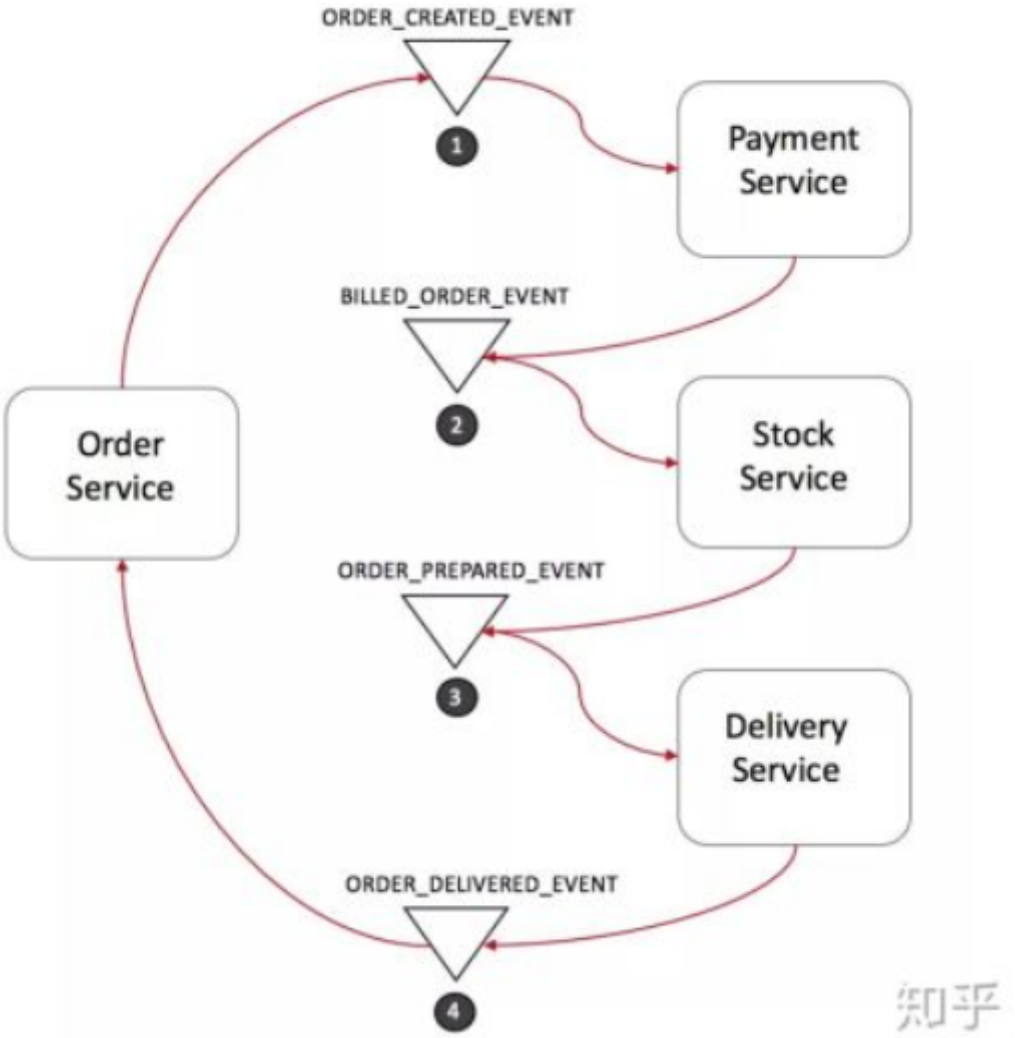
在基于事件的方式中，第一个服务执行完本地事务之后，会产生一个事件。其它服务会监听这个事件，触发该服务本地事务的执行，并产生新的事件。

我们继续以订单流程为例，说明一下该模式。

假设一个完整的订单流程包含了如下几个服务：

- Order Service：订单服务
- Payment Service：支付服务
- Stock Service：库存服务
- Delivery Service：物流服务

采用基于事件的saga模式的订单处理流程如下：



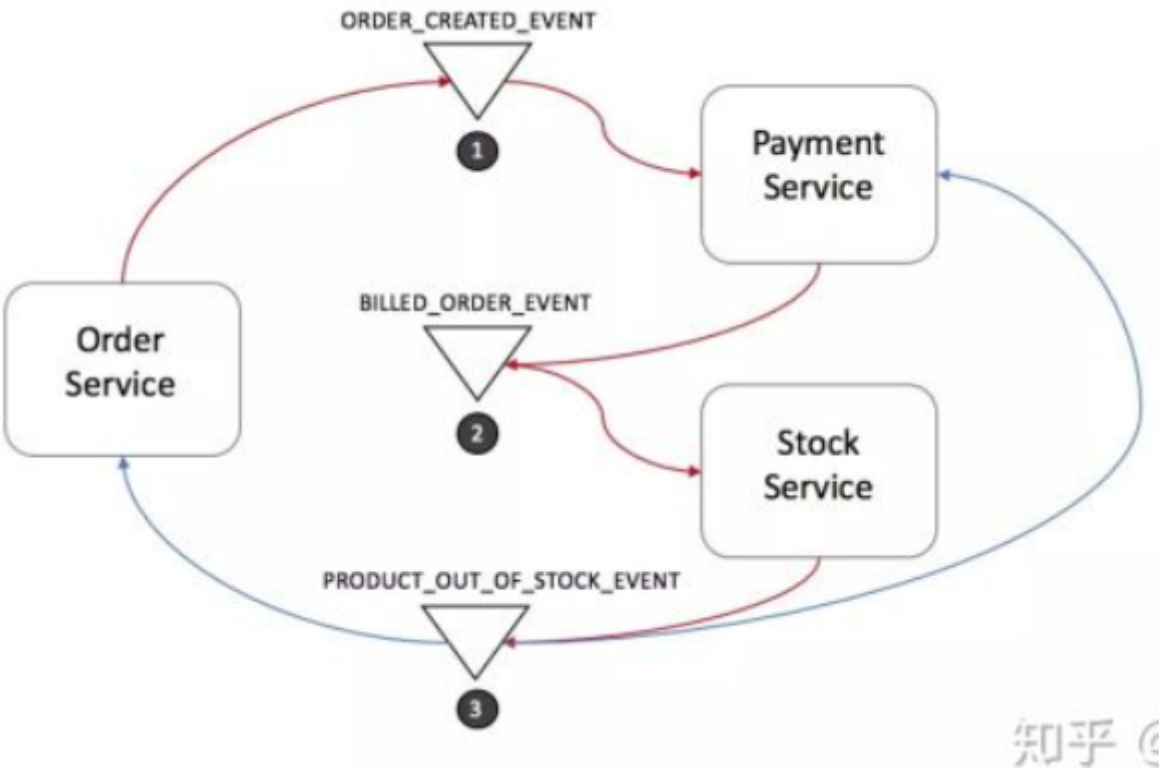
1. 订单服务创建一笔新订单，将订单状态设置为"待处理"，产生事件ORDER_CREATED_EVENT。
2. 支付服务监听ORDER_CREATED_EVENT，完成扣款并产生事件BILLED_ORDER_EVENT。
3. 库存服务监听BILLED_ORDER_EVENT，完成库存扣减和备货，产生事件ORDER_PREPARED_EVENT。
4. 物流服务监听ORDER_PREPARED_EVENT，完成商品配送，产生事件ORDER_DELIVERED_EVENT。
5. 订单服务监听ORDER_DELIVERED_EVENT，将订单状态更新为"完成"。

在这个流程中，订单服务很可能还会监听BILLED_ORDER_EVENT，ORDER_PREPARED_EVENT来完成订单状态的实时更新。将订单状态分别更新为"已经支付"和"已经出库"等状态来及时反映订单的最新状态。

该模式下分布式事务的回滚

为了在异常情况下回滚整个分布式事务，我们需要为相关服务提供补偿操作接口。

假设库存服务由于库存不足没能正确完成备货，我们可以按照下面的流程来回滚整个Saga事务：



1. 库存服务产生事件PRODUCT_OUT_OF_STOCK_EVENT。
2. 订单服务和支付服务都会监听该事件并做出响应：
    a. 支付服务完成退款。
    b. 订单服务将订单状态设置为"失败"。

基于事件方式的优缺点

优点：简单且容易理解。各参与方相互之间无直接沟通，完全解耦。这种方式比较适合整个分布式事务只有2-4个步骤的情形。

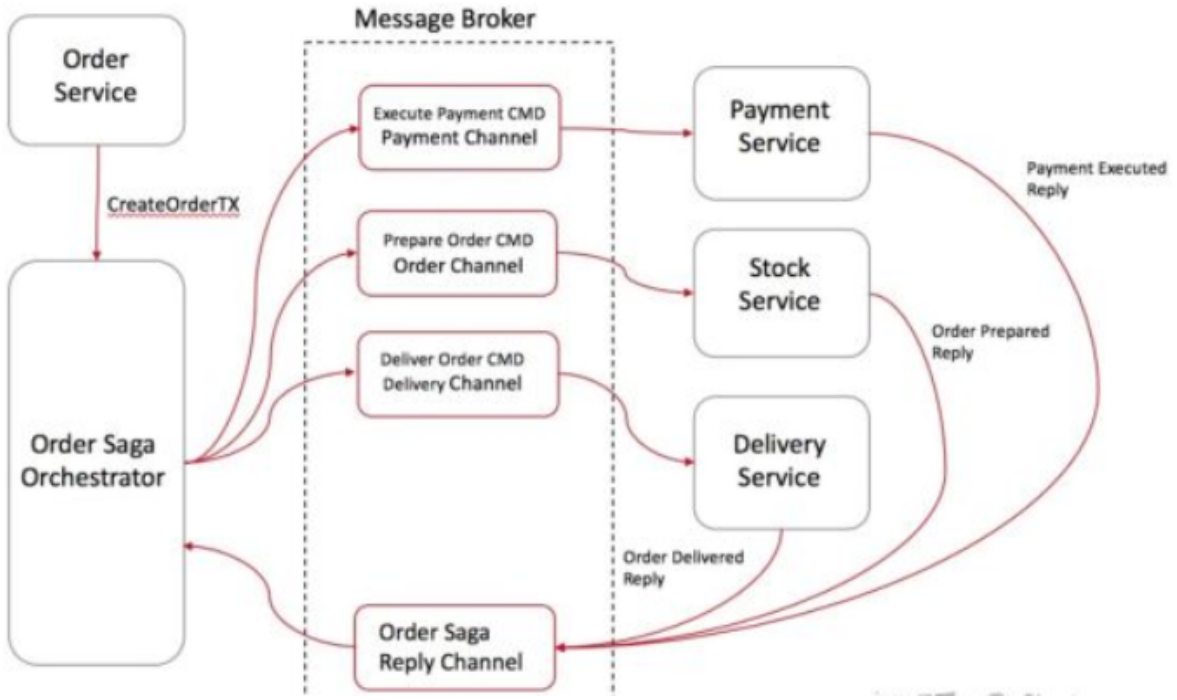
缺点：这种方式如果涉及比较多的业务参与方，则比较容易失控。各业务参与方可随意监听对方的消息，以至于最后没人知道到底有哪些系统在监听哪些消息。更悲催的是，这个模式还可能产生环形监听，也就是两个业务方相互监听对方所产生的事件。

接下来，我们将介绍如何使用命令的方式来克服上面提到的缺点。

基于命令的方式

在基于命令的方式中，我们会定义一个新的服务，这个服务扮演的角色就和一支交响乐乐队的指挥一样，告诉各个业务参与方，在什么时候做什么事情。我们管这个新服务叫做协调中心。协调中心通过命令/回复的方式来和Saga中其它服务进行交互。

我们继续以之前的订单流程来举例。下图中的Order Saga Orchestrator就是新引入的协调中心。



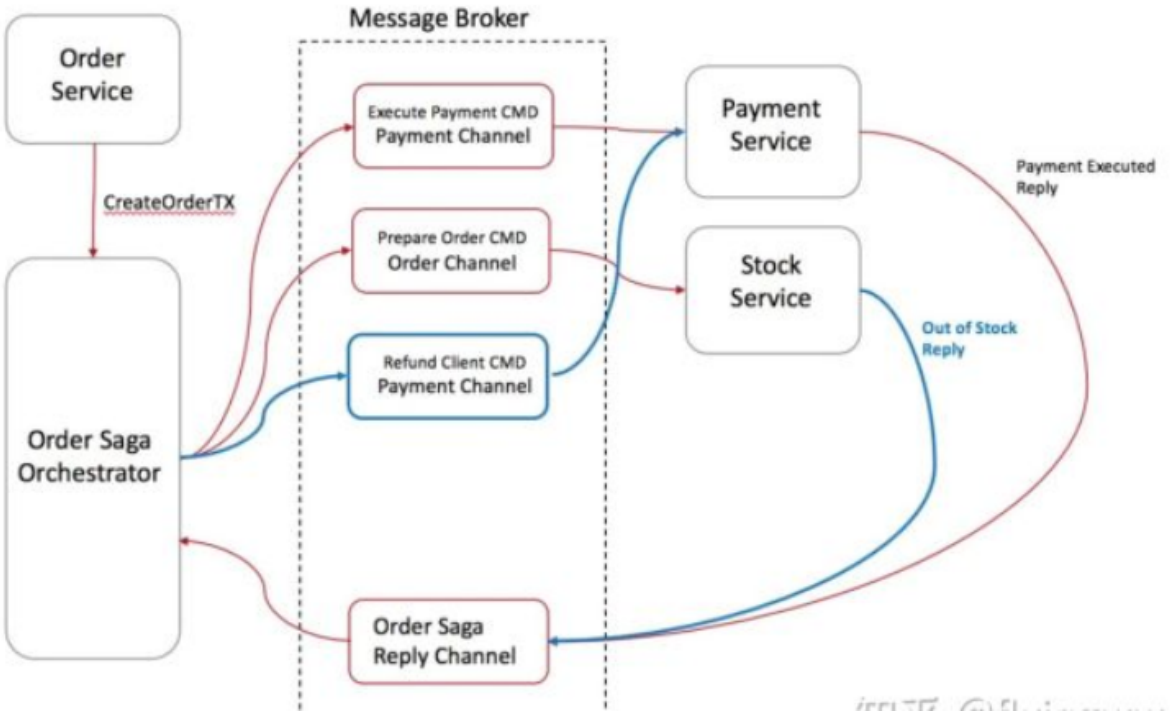
1. 订单服务创建一笔新订单，将订单状态设置为"待处理"，然后让Order Saga Orchestrator（OSO）开启创建订单事务。
2. OSO发送一个"支付命令"给支付服务，支付服务完成扣款并回复"支付完成"消息。
3. OSO发送一个"备货命令"给库存服务，库存服务完成库存扣减和备货，并回复"出库"消息。
4. OSO发送一个"配送命令"给物流服务，物流服务完成配送，并回复"配送完成"消息。
5. OSO向订单服务发送"订单结束命令"给订单服务，订单服务将订单状态设置为"完成"。

OSO清楚一个订单处理Saga的具体流程，并在出现异常时向相关服务发送补偿命令来回滚整个分布式事务。

实现协调中心的一个比较好的方式是使用状态机（Sate Machine）。

该模式下分布式事务的回滚

该模式下的回滚流程如下：



1. 库存服务回复OSO一个"库存不足"消息。
2. OSO意识到该分布式事务失败了，触发回滚流程：
    a. OSO发送"退款命令"给支付服务，支付服务完成退款并回复"退款成功"消息。
    b. OSO向订单服务发送"将订单状态改为失败命令"，订单服务将订单状态更新为"失败"。

基于命令方式的优缺点

优点：

1. 避免了业务方之间的环形依赖。
2. 将分布式事务的管理交由协调中心管理，协调中心对整个逻辑非常清楚。
3. 减少了业务参与方的复杂度。这些业务参与方不再需要监听不同的消息，只是需要响应命令并回复消息。

4. 测试更容易（分布式事务逻辑存在于协调中心，而不是分散在各业务方）。
5. 回滚也更容易。

缺点：
一个可能的缺点就是需要维护协调中心，而这个协调中心并不属于任何业务方。

Saga模式小窍门

1. 给每一个分布式事务创建一个唯一的Tx id。这个唯一的Tx id可以用来在各个业务参与方沟通时精确定位哪一笔分布式事务。
2. 对于基于命令的方式，在命令中携带回复地址。这种方式可以让服务同时响应多个协调中心请求。
3. 幂等性。幂等性能够增加系统的容错性，让各个业务参与方服务提供幂等性操作，能够在遇到异常情况下进行重试。
4. 尽量在命令或者消息中携带下游处理需要的业务数据，避免下游处理时需要调用消息产生方接口获取更多数据。减少系统之间的相互依赖。

总结

上面订单流程中的最后一个步骤，物流服务，基本上已经体现了Saga模式的特点。那就是Saga非常适合用来处理时间跨度比较长的分布式事务问题。同时，对于分布式事务参与方的完成时效性没有要求。

要在实际项目中使用Saga模式，还有一个重要问题需要解决。如何在本地事务中可靠地产生/发送一个事件。对于基于事件的方式，服务参与方在本地事务执行完毕后，需要能确保在当前事务中可靠地产生一个事件， 来触发后续服务中本地事务的执行；而对于基于命令的方式，也需要解决命令和回复生成方式的可靠性问题。

参考：https://zhuanlan.zhihu.com/p/95852045