原文地址：http://www.360doc.com/content/17/1121/14/203871\_705844975.shtml

**什么是Saga**

Saga是什么Saga是什么？Saga的定义是“长时间活动的事务”(Long Lived Transaction，后文简称为LLT)。但Saga时间跨度并不重要，有时候甚至只有几秒钟。

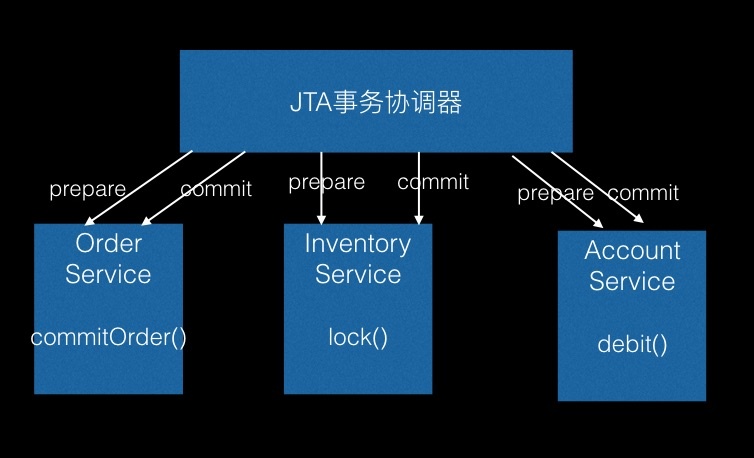
Saga往往由多个外部子事务构成，需要通过多次外部系统的消息交互，才能将整体事务从开始迁移到结束状态

ACID事务 - 要么一起成功，要么一起失败。但是LLT的话，直接ACID，消耗就太大了，想象一下我们预定一张机票和酒店，在酒店人员确认房间信息前，我们前一个机票必须一直都处于占位锁定但不能确认出票的状态，机票供应商可不允许你这样玩。

Saga采用了BASE（Basic Availability, Soft, Eventual consistency）事务的方式，来避免消耗大量资源的同步、锁定。

举例：电商下单，需要冻结商品库存（将可销售库存-1，已销售库存+1），然后扣取用户的帐户余额 ，最终完成订单。 库存不足、帐户余额不足都将导致订单失败。这里库存和支付模块都是单独的服务

如果要实现ACID，控制器统一向三个服务发起prepare的请求，如果都确认，就可以同时事务提交（请注意这里订单一旦生成就已经是commited状态了）。提交过程中所有的资源是锁定等待中的，如果账务扣款慢了（比如银行接口反应慢了），资源就会一直等待锁定，直到事务超时为止，这样对系统资源的消耗是巨大的。



如果采用BASE事务，将大事务拆为5个local transaction(T1-T5)，并为这些事务准备了冲正操作

T1：Order服务先完成开启订单，订单状态Opened

T2：Inventory服务完成锁定库存

T3：Order服务变更订单状态为Prepared

T4：Account服务扣款

T5：Order服务变更订单状态为Committed

其中每步完成都会提交事务。请注意这里的订单多了两个中间状态Opened和Prepared，最后才到Committed。和ACID相比，BASE不能轻易的回滚，只能通过补偿（Compensating）操作达到最终一致性。比如在T4扣款阶段，如果发现用户账户余额不足，交易失败。那么就要执行前三步的冲正操作(C3、C2、C1)

C3：将订单状态改为Opened

C2：将库存解锁

C1：将订单置为废弃状态 Cancelled

注：这里的冲正的概念，英文术语是Compensating（补偿），和原来的事务回滚Roll back不一样，回滚是可以撤销变更回到原有的状态的。ACID下如果库存或者余额不足，订单不会生成，直接回滚。BASE下即使最终订单失败取消，订单还是会生成，对世界的改变已经不可撤销了。

**Saga特点**

SEC

我们有两种方式来处理Saga中的业务流程和失败补偿

做法一：让每个服务给后续步骤发送消息

做法二：通过一个统一的执行协调器（SEC）来完成

我们倾向于有一个统一的协调器来完成这个操作。

Saga本身是不包含业务逻辑的，Saga更倾向于“Process”而不是“Logic”。

多实例

SEC的每个实例只包含单个Saga的状态，这就意味着如果你有多个订单，就会启动多个SEC。

Event Handling

SEC需要用到Event Handling机制。让Saga的SEC实例自己来订阅属于自身的Event

持久化

SEC的事务可能会持续很长时间，为了考虑停机重启等场景，我们会将Saga持久化