# 分布式事务

1. 本地事务在分布式事务下的问题
   1. 其实就是本地事务只能控制住自己，反正他妈的一旦出现一个微服务里面调两个微服务的情况就会产生这种问题，因为一旦远程调用出现了网络延迟，或者宕机，他直接上层那个服务还好，能回滚不，但是另外中间那一个呢，肯定回滚不了啊
   2. 说白了最大的问题就是：网络抖动和宕机
2. CAP理论
   1. C 一致性（强一致性）
      1. 假设现在ABC三台机器，现在发了一个请求，A接收到了，BC也得同步一下
   2. A 可用性
      1. 假如我现在有三台机器是吧，有一台挂了，那么这整个集群要是还能响应请求，那就是可用的
   3. P 分区容错性
      1. 机器A在华北地区，B在华东，他们不在同一网络下，就肯定会发生这个网络故障啊什么的就导致两台机器之间无法通信
   4. CAP三者是不可兼得的，只能是CP或者AP
   5. 一致性的保证可以由这个raft算法保证，它的节点呢是有三个状态的，随从和候选，领导。一开始都是随从，如果有一个挂了，另外两个就会开始选举，这个过程大概就是这样的，不是很了解
3. BASE理论
   1. BASE理论其实是CAP的一个延伸，因为现在主要还是保证可用性嘛，所以大部分还是CP系统，但是有些业务还是需要保证一致性的，那么就可以抛弃强一致性，可以来个弱一致性（也就是利用数据副本）
   2. 可以从三个方面讲
      1. 基本可用：意思就是假如宕机了，那么这个功能我就不算了，给他提供一个降级的页面（比如双11你把淘宝冲垮了，人家会搞一部分请求去一个降级的页面呀，但是服务还是可用的），还有就是响应慢一点点也没关系不，牺牲一点响应速度
      2. 软状态：就这次投简历来说，BOSS软件他上传简历，上传不成功，但是呢，它也不崩，一直在转圈圈，这个我感觉就可以理解为是软状态
      3. 最终一致性：就好比一个下单业务，下单一般会生成一个订单保存是吧，假设我现在下单，它需要调用扣减库存，然后调用扣积分两个服务，万一到扣积分的时候崩了，那么下单是会收到的不，进行回滚，订单就不会保存了，那么库存呢，是不是扣了呀。这个时候就可以使用弱一致性，即使扣了库存，你给用户提示个下单失败也没关系嘛，我过一段时间再去恢复就行了
4. 分布式事务常见解决方案
   1. 2pc模式也称为XA二阶段提交协议
      1. 怎么说呢，不适合高并发情况下使用，性能不好，因为他是基于ACID的强一致性（刚性事务），但是比较简单咯
   2. 柔性事务-TCC事务补偿型方案
      1. 就是他妈的让程序员一个业务拆成三个部分，分别是 预备状态try（准备数据），confirm（提交），cancel（回滚），它的过程大概是这样子的，它会有一个业务管理器来管理
      2. 第一阶段，它会让业务都去执行try，都成功之后嘞，就进入第二阶段，进行提交，一旦有一个地方失败了呀，就会调用那个回滚方法
      3. 性能也不咋地
   3. 柔性事务-最大努力通知型
      1. 比如说我下单服务调用库存扣减，再调用扣除积分，如果说积分扣除失败，那么这个订单服务就会发一个消息给mq，让那些服务都订阅mq，让他们自个同步去，这样就可以承受大并发
   4. 柔性事务-可靠消息+最终一致性方案
      1. 他这个就跟上面差不多，但是他这个可靠消息，mq必须确认收到订阅者这个消息，才罢休，应该是这样
5. seata 控制分布式事务 AT模式
   1. 适用于并发不高的，就有个农产品保存模块，其中调用了两个其他微服务模块的保存，这就很适合 seata
   2. 实现每一个微服务都会创建一个 undo-log表，回滚日志表
   3. 安装一个事务协调器：seata-server
   4. 将其整合
      1. 导入依赖 版本是 0.7.1
      2. 修改它的一些配置：registry.conf ：注册中心需要配置成nacos
      3. 然后所有需要使用这个分布式事务的微服务都得配置一下数据源，让seata来代理
      4. 每个服务还得导入那两个文件
      5. 给分布式大事务就是起初调用的那个，加一个@GloablTransactional
      6. 小事务还是@Transactional