微服务一个服务就是一个需求就是一个项目，这个项目可以开启多个负载均衡

所以一个服务可以包含多个进程

单体应用修改一部分就得重新部署，微服务只需要部署哪个服务

但是也不绝对，其他服务依赖于这个服务，也可能得相应的修改部署

微服务通讯需要http或rpc，需要一定的代价。没单体应用调用那么方便

微服务拆分，在设计服务的时候某种程度上耦合度会减少

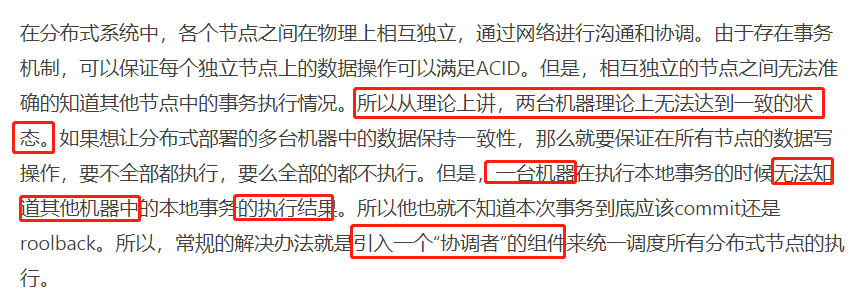
微服务服务间尽可能大粒度的调用，传入比较多的数据，得到比较多的响应（需要连接资源）

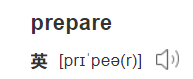
而不是传统的方法间多调用几次

微服务的不同服务间可以用不同的语言实现

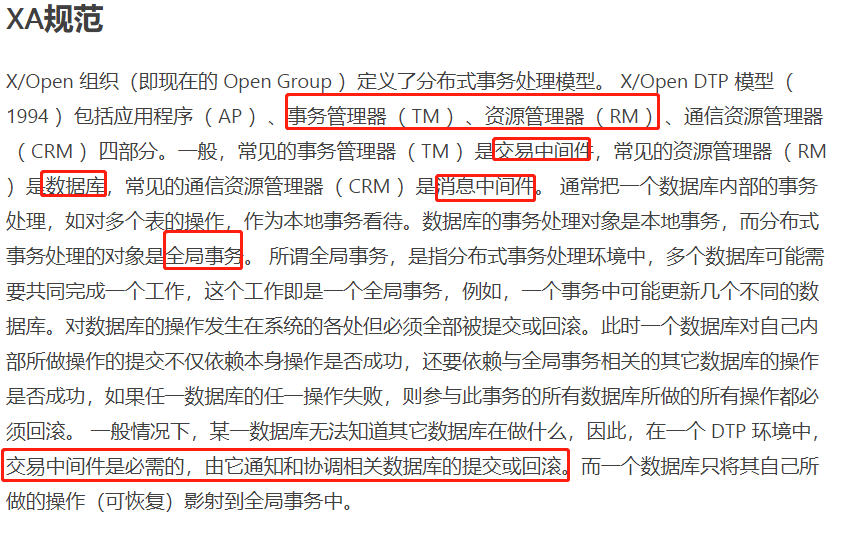


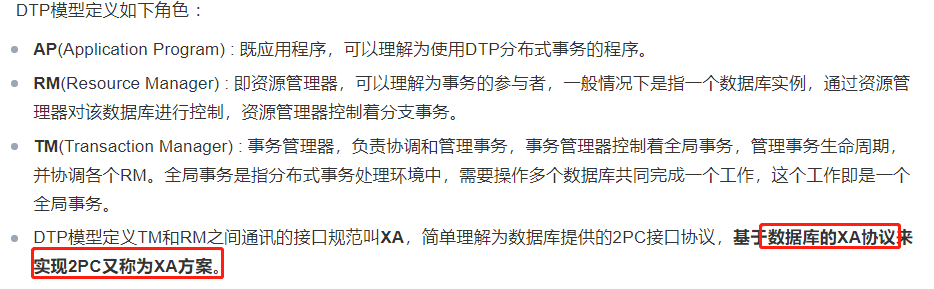
去中心话倡导多数据库，多语言开发

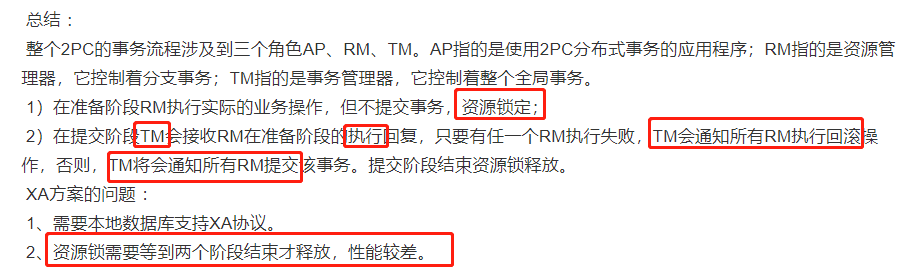




2pc prepare commit





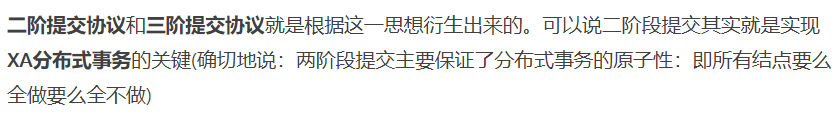


准备阶段 ：吃饭前，老板要求张三付款，张三准备付款。老板要求李四付款，李四准备付款。

提交阶段 ：张三付款，李四准备，老板出票，两人拿票纷纷落座就餐。

准备付款，还没付，资源锁定。

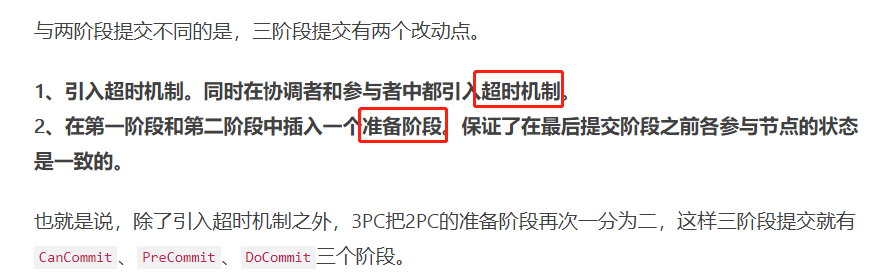
提交执行付款，一个没付那就出票失败



XA是协议是思想，pc是最终实现

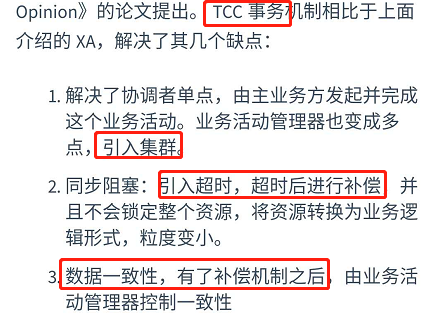


<https://www.jianshu.com/p/30a18e4ef16e>



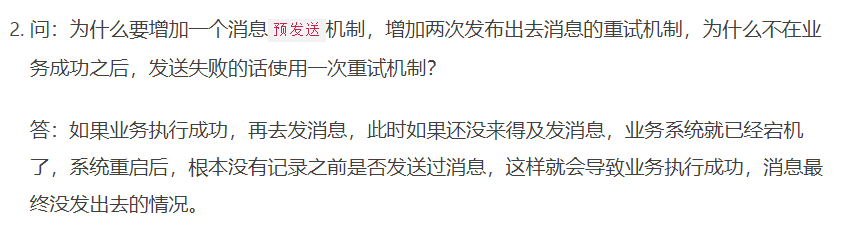




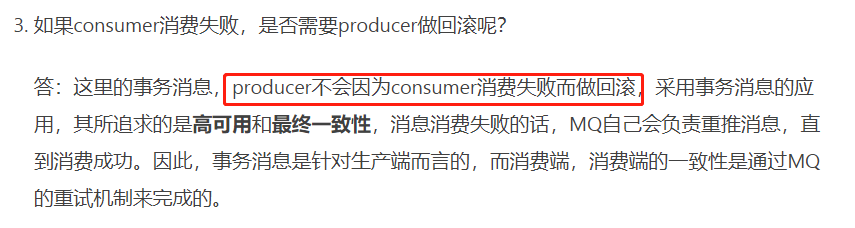


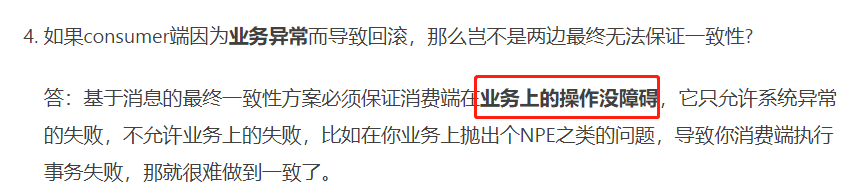
TCC代码复杂性也会变高

消息队列事务最终一致性一个很重要的点就是回查机制



所以要有回查状态，查看事务是否执行成功





所以mq的事务只能支持生产者的回滚，而TCC补偿机制两边都能回滚

2PC需要控制资源，不适用于微服务

2PC强一致性，要么一起成功，要么一起失败间隔短

TCC和消息队列，一遍成功了，另一遍或许好久后才成功,失败慢慢补偿回去

TCC的应用范围更广，不一定是事物关系数据库，也可能操作的KV数据库，文档数据库，粒度也可以随着具体业务灵活调整，性能更好

服务任何时候都可以导致失败，得做好监控，报警，退级什么的（短路，熔断）

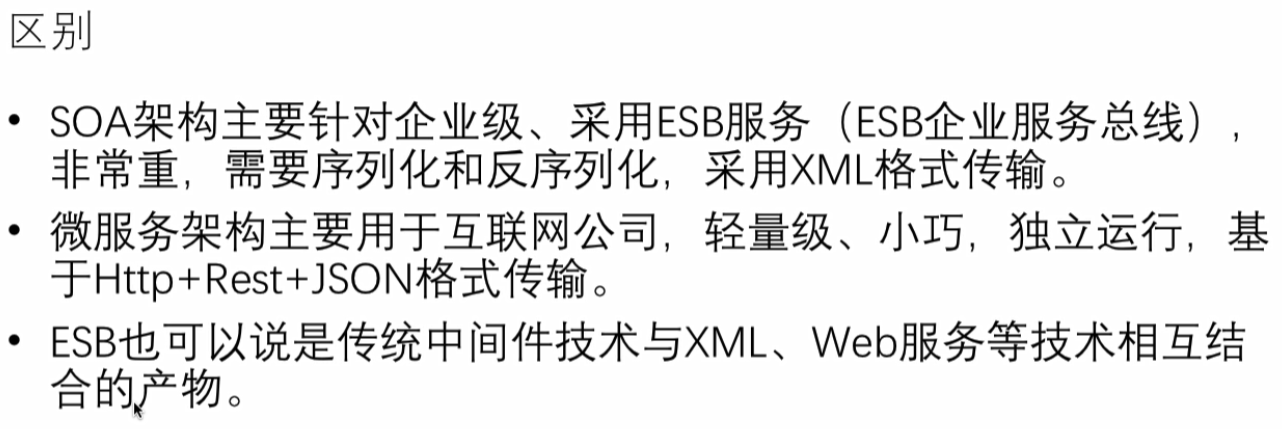
一个服务失败不会影响全局

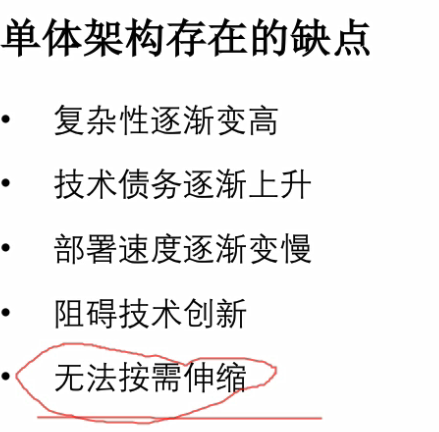
单体应用，服务间调用方便，没有网络上的问题，出错几率小，部署较为方便，事务的话控制方便

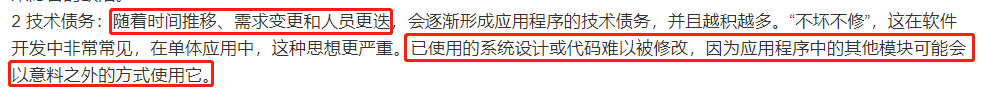
微服务边界比较难控制，其他服务的重构也不方便,接口的改变需要协调。测试，定位问题也比较困难

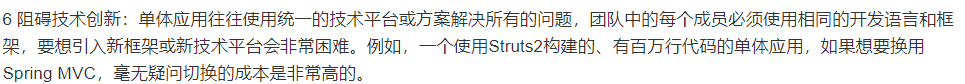
就是你改了你需求的接口，其他团队可能也得修改代码。需要更多的沟通

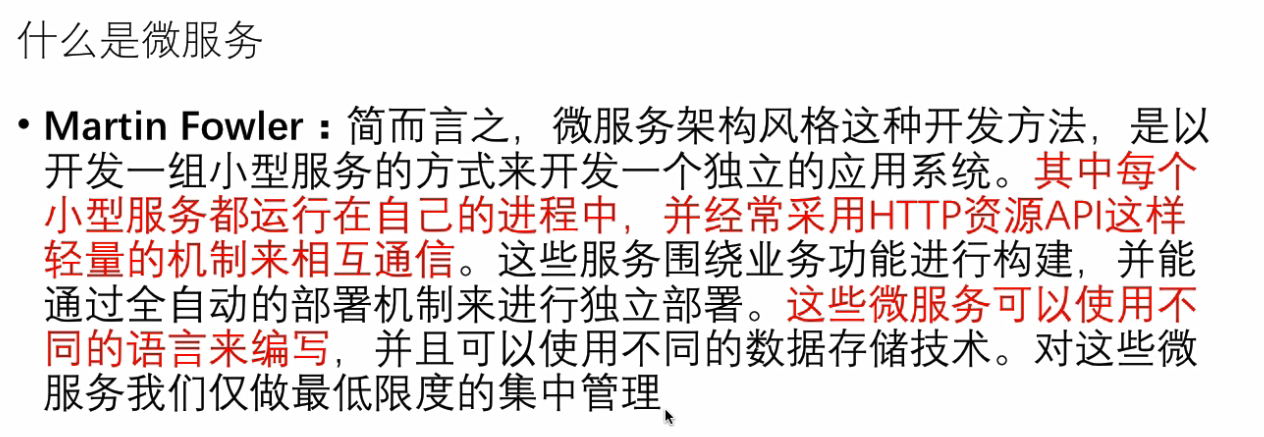
服务与服务之间是有调用关系的

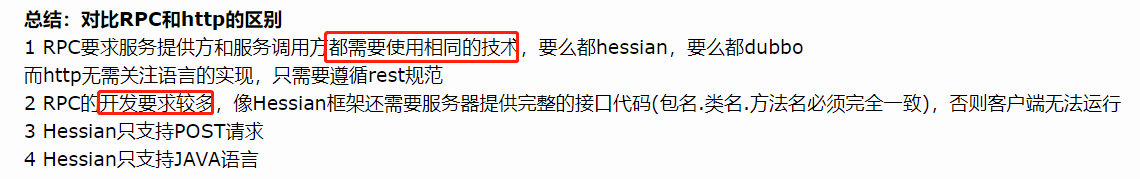


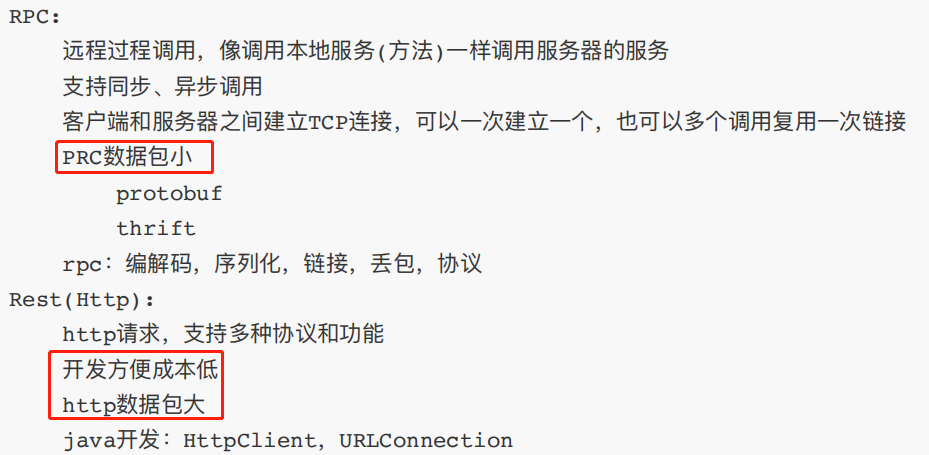




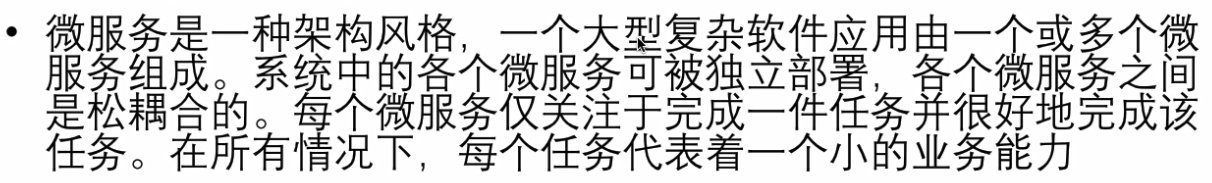


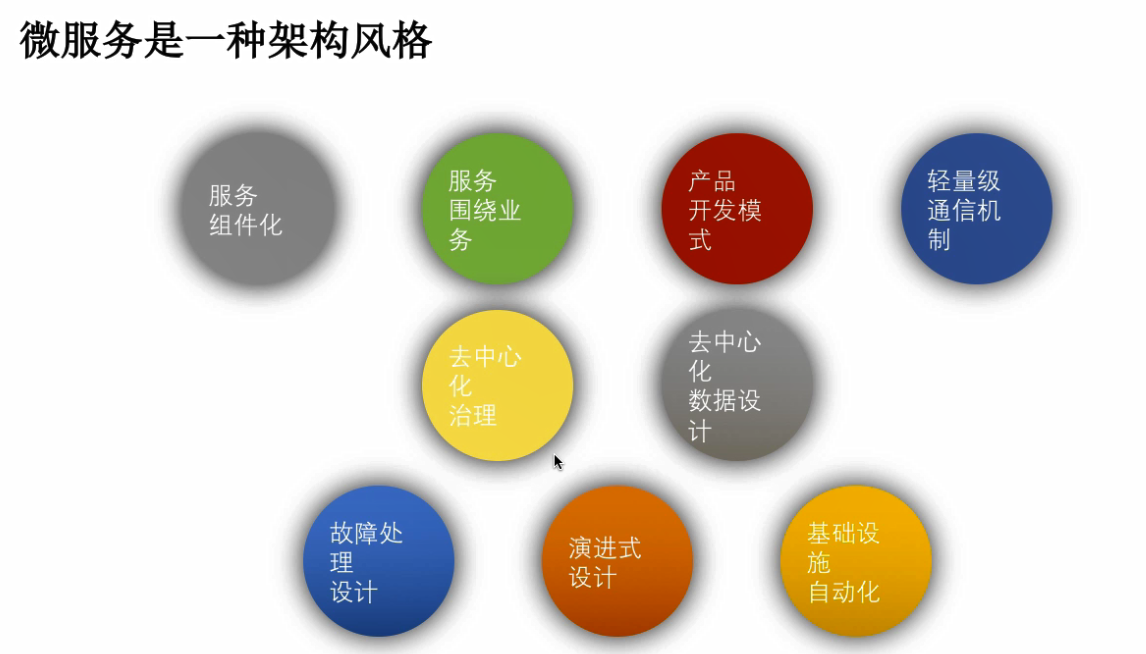


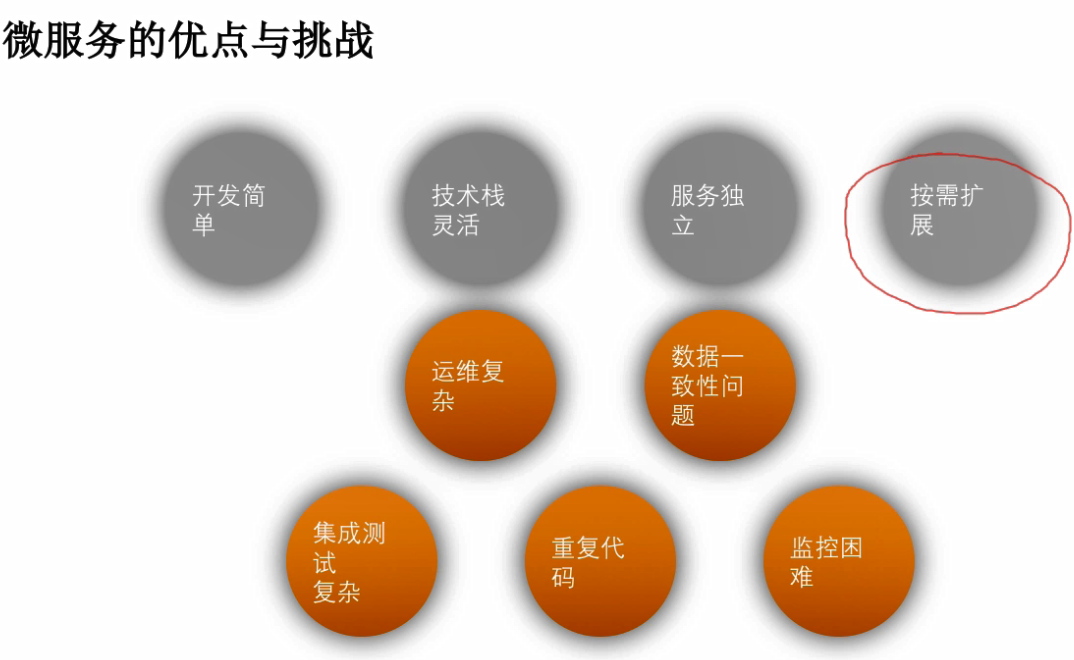




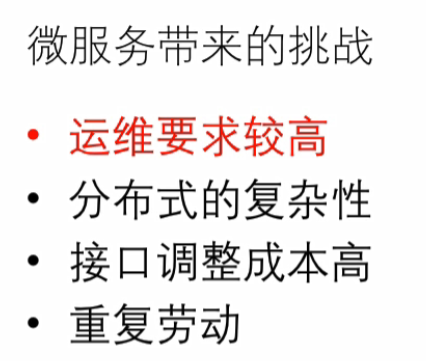
rpc包小，传输速度快，开发复杂，需要双方都用相同的技术

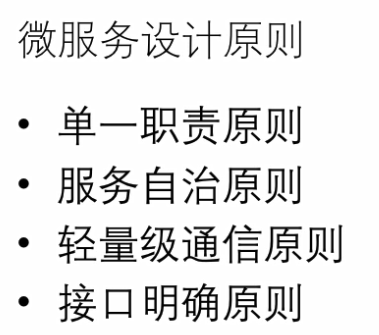






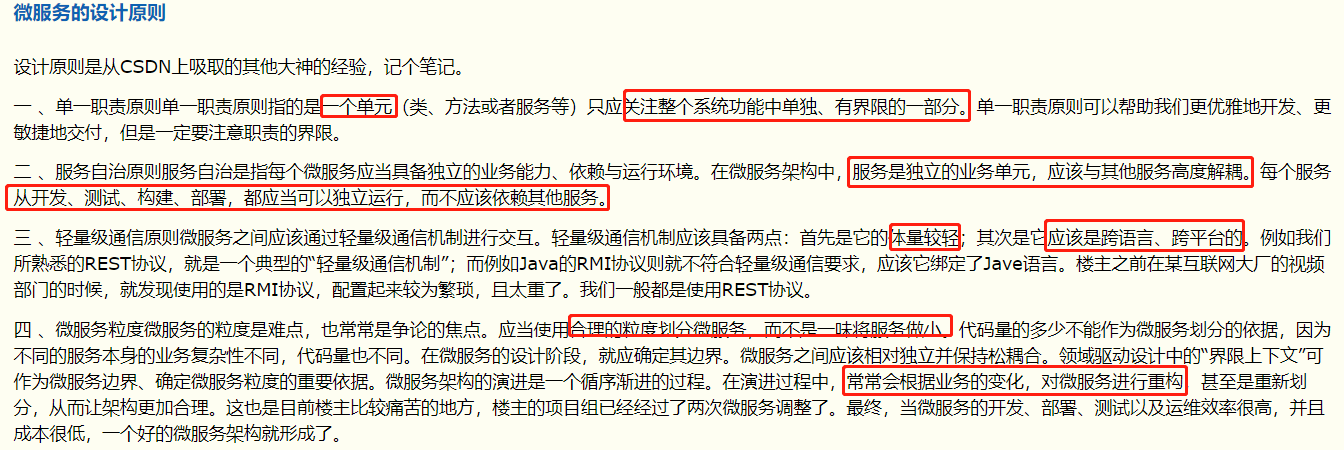
重复代码，可以用jar包解决，但是改动大的话，得通知更新jar包，需要更新成本



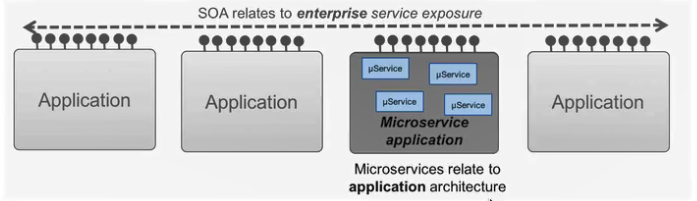


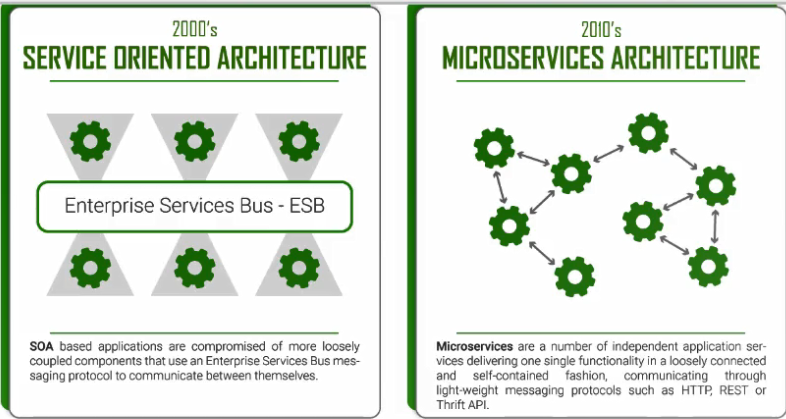
基于session和cookie都可以实现事务，可以认为，session是有状态的，而cookie是无状态的

服务的设置最好是无状态的，集群时访问不同机器时，有状态数据再第一台机器，那么访问第二台机器时就拿不到数据了



soa的范围（企业）更大，微服务范围（应用）小





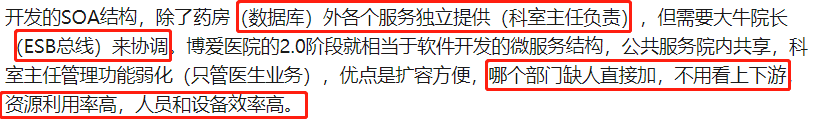
soa 应用服务的可重用性，微服务解耦

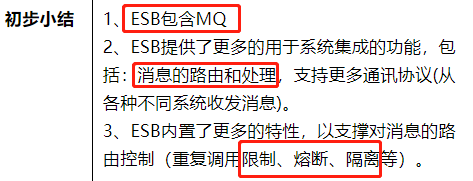
soa共享数据存储，微服务每个服务都有自己的数据库

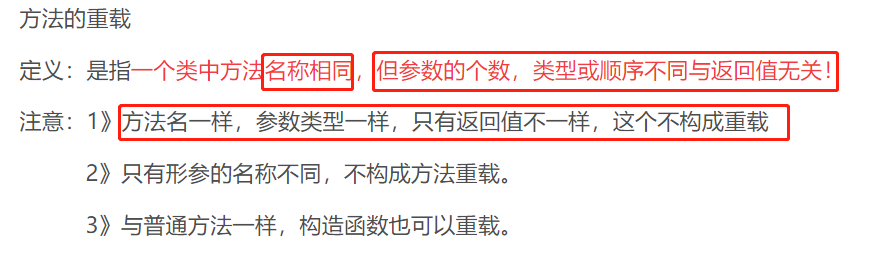
微服务比如订单服务，商品服务

soa相当于一个商品应用里有订单服务，商品服务，现在有多个应用通过esb连接

微服务架构首先要关注的是边界







返回值和方法类型一样，无关