# 微服务搭建方案

## springcloud + k8s搭建微服务

平台集成Service Mesh功能，客户搭建起微服务来很简单

# 分布式事物

分布式事物的存在，就是解决数据不一致的情况。分布式系统中有这么一个广为流传的理论：CAP 定理对于一个分布式计算系统来说，不可能同时满足以下三点：

•一致性（Consistency）

•可用性（Availability）

•分区容错性（Partition tolerance）

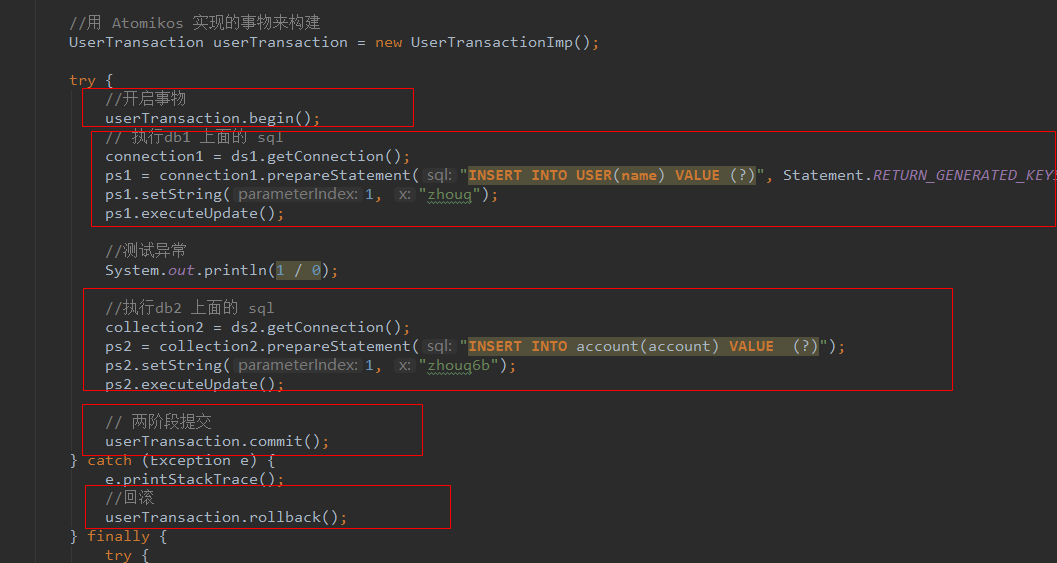
由于分区容错性（P）是必须保证的，那么我们分布式系统就更多是在一致性（CP） 和可用性（AP）上做平衡了，只能同时满足两个条件。其实，大家想想，ZK 是不是就是严格实现了 CP ，而 Eureka 则是保证了 AP。其实分布式事物强调的就是一致性。

## 几种分布式事物解决方案

**2PC**

在说 2PC 之前，我们先了解一下 XA规范 是个什么东西？

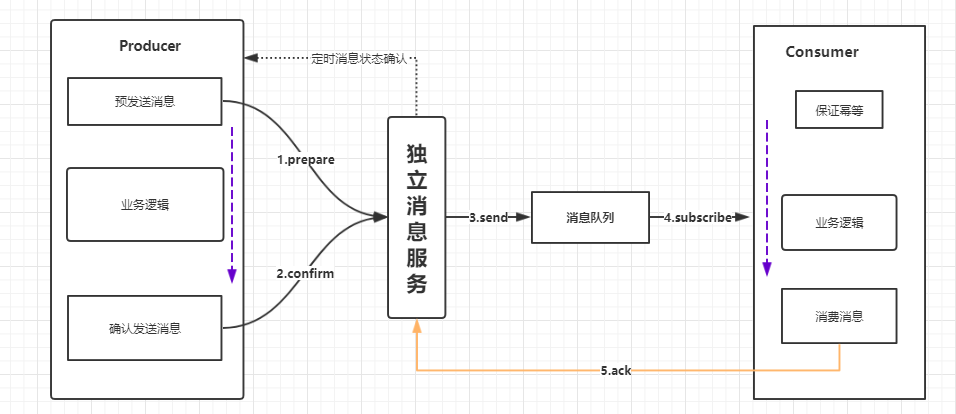
XA规范描述了全局的事务管理器与局部的资源管理器之间的接口。XA规范的目的是允许多个资源（如数据库，应用服务器，消息队列，等等）在同一事务中访问，这样可以使ACID属性跨越应用程序而保持有效。



XA 使用 两阶段提交（2PC） 来保证所有资源同时提交或回滚任何特定的事务。（2PC即两个事物分别预提交，再确认提交，如果报错则回滚。）

**可靠消息最终一致性方案来解决分布式事物的问题**

这个方案，就有消息服务中间件角色参与进来了。



后面的系统，一定要处理 幂等，重试，日志这几个东西。

**TCC 方案**

TCC 的全程分为三个阶段，分别是 Try、Confirm、Cancel：

1.Try阶段：这个阶段说的是对各个服务的资源做检测以及对资源进行锁定或者预留

2.Confirm阶段：这个阶段说的是在各个服务中执行实际的操作

3 .Cancel阶段：如果任何一个服务的业务方法执行出错，那么这里就需要进行补偿，就是执行已经执行成功的业务逻辑的回滚操作

还是以转账的例子为例大概思路就是这样的：

1.Try 阶段：先把A 银行账户先冻结 1 块，B银行账户中的资金给预加 1 块。

2.Confirm 阶段：执行实际的转账操作，A银行账户的资金扣减 1块，B 银行账户的资金增加 1 块。

3.Cancel 阶段：如果任何一个银行的操作执行失败，那么就需要回滚进行补偿，就是比如A银行账户如果已经扣减了，但是B银行账户资金增加失败了，那么就得把A银行账户资金给加回去。