可靠消息服务设计

# 概述

消息队列中间件是分布式系统中重要的组件，主要解决应用耦合，异步消息，流量削锋等问题。实现高性能，高可用，可伸缩和最终一致性架构。是大型分布式系统不可缺少的中间件。

目前在生产环境，使用较多的消息队列有ActiveMQ，RabbitMQ，ZeroMQ，Kafka，MetaMQ，RocketMQ等。

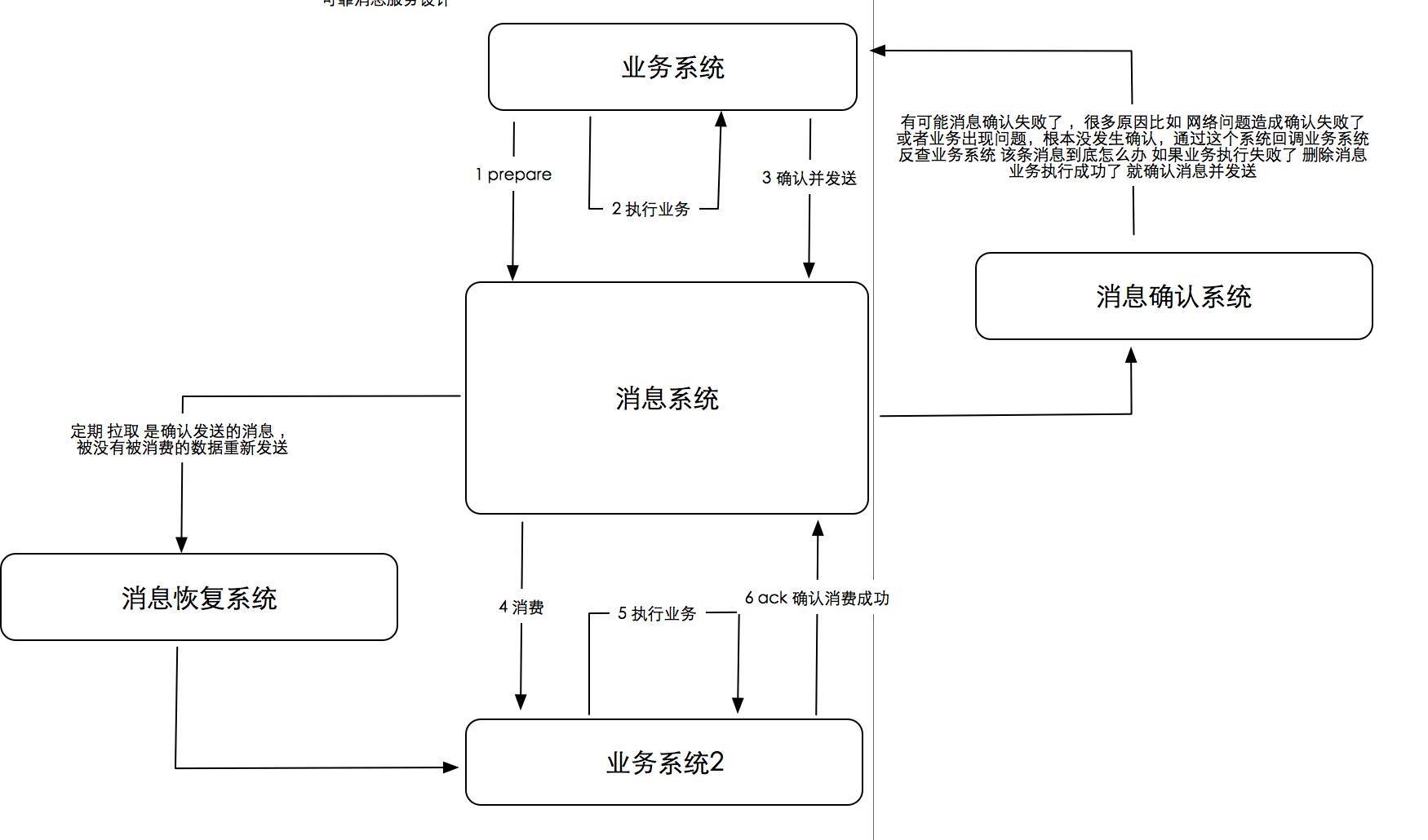
举个例子，Bob向Smith转账，那我们到底是先发送消息，还是先执行扣款操作？

好像都可能会出问题。如果先发消息，扣款操作失败，那么Smith的账户里面会多出一笔钱。反过来，如果先执行扣款操作，后发送消息，那有可能扣款成功了但是消息没发出去，Smith收不到钱。

可靠消息最终一致方案主要是借助MQ消息系统来进行事务控制。当然简单的使用MQ显然是达不到事务控制的目的的，因为你无法把本地的数据库操作与消息发送操作进行统一事务控制。因此需要一定的巧妙设计。

如果分布式事务在系统中只是偶尔遇到的情况，可以采用相对简单的处理方式，将要发送的消息，先保存到本地数据库，这样就可以与其它业务操作统一通过本地事务控制，然后通过定时器或者独立的线程轮询本地的消息表，将数据发送到MQ中间件供消费者消费。这实际上将操作分成了两部分，首先将消息数据保存到了本地，与其它操作要么一起成功，要么一起失败回滚，保证了一致性。然后通过定时器或者独立的线程进行消息确认、消息查询、消息恢复等操作，确保将数据发给MQ，利用MQ的机制确保消费者可以收到消息。这种方式的水平比较低，与具体业务场景绑定，实时性差，且独立线程消耗资源。

# 可靠消息设计



开始执行 比如：

try{

if(prepare()) { //预发送阶段

doService(); //执行业务逻辑

updateMsgStatus();//更新消息为确认状态

}

}

1 预发送消息，try阶段，如果预发送消息失败了，业务还未执行，所以系统A，B还是一致性的，不需要处理。

2 预发送消息成功了，开始执行业务逻辑。执行成功 更新预发送消息转态为确认发送。如果 此时 业务逻辑执行失败了，那预发送消息就不会跟新状态，此时消息确认系统就开启工作，到业务系统1上回查此消息状态，此时发现业务执行失败了，就更新预发送状态至失败状态。

3 如果此时 业务执行成功了消息也被更新成确认发送了 那就ok 完美。如果消息更新失败，还是由消息确认系统回查转态 更新此消息被删除状态还是确认发送状态。

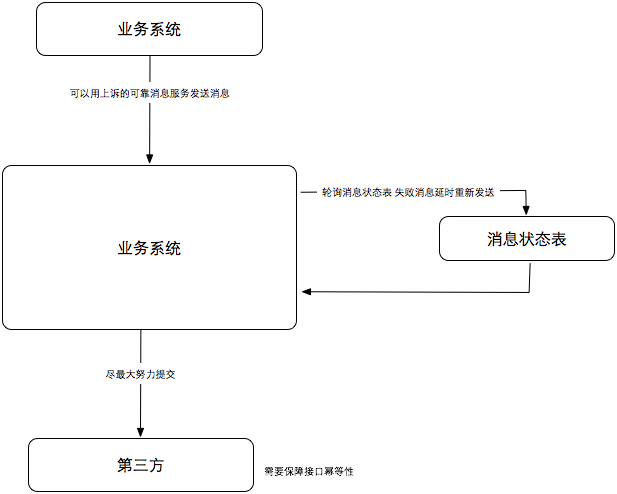
4 消息者开始消费

* 比如消费失败了，此时产生不一致， 消息恢复系统检测消息状态，重新发送消息
* 如果执行业务失败了，此消息也就不会被确认了，还是由消息恢复系统检测消息状态，重新发送消息
* 如果ask失败了，还是以上逻辑重新发送上诉重新发送当然有次数限制，不能一直发送，超过最大次数就要进入死信队列，等待人工干预了
* ask成功，消息也就成功消费了，完美，解决了消息可靠服务

# 努力提交

这个比较简单 ，将失败的消息重复提交，实时性比较弱的一些场景，确保消息推送成功。

比如交易完成推送第三方消息。 此时可以使用努力提交



# 附录

<https://blog.csdn.net/zsh2050/article/details/78023215>

<http://www.sohu.com/a/200535613_610671>

https://blog.csdn.net/wuzhiwei549/article/details/79788100