架构实战营模块6

第2课：微服务架构陷阱与挑战

李运华

前阿里资深技术专家（P9）

教学目标

极客时间 训练营

1. 理解微服务架构常见的陷阱
2. 掌握微服务架构带来的挑战和应对方法

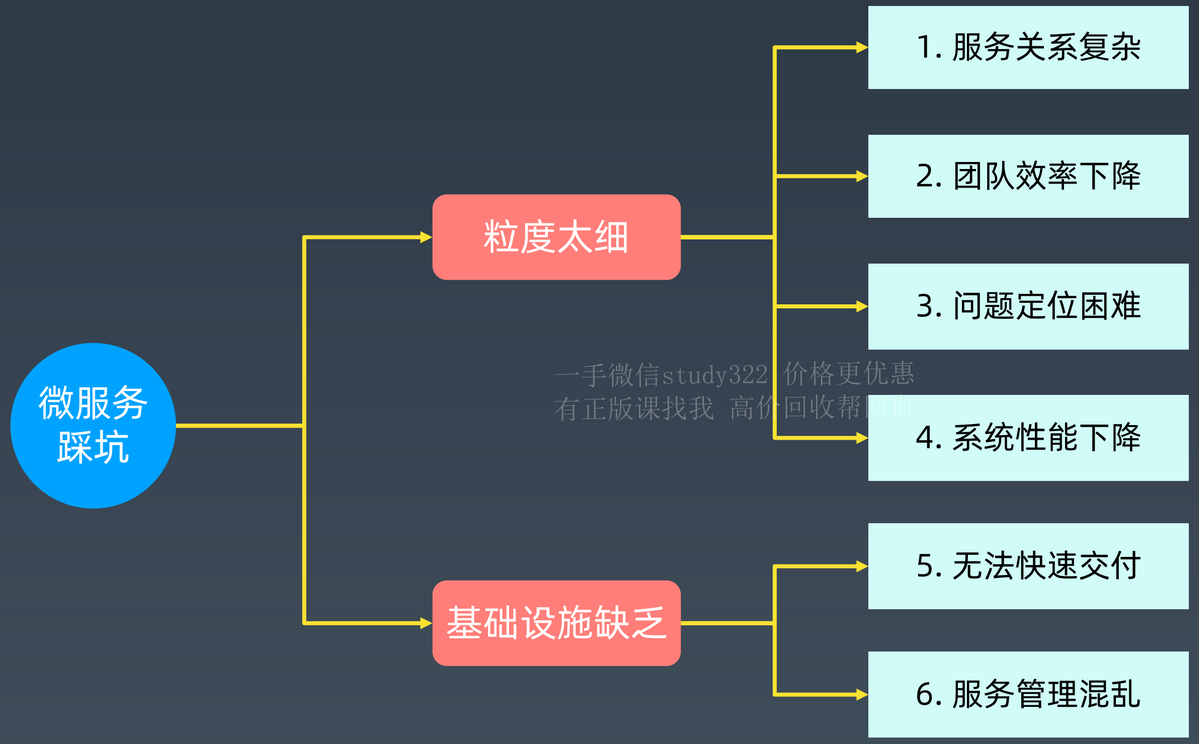
细节是魔鬼！

极客时间 训练营

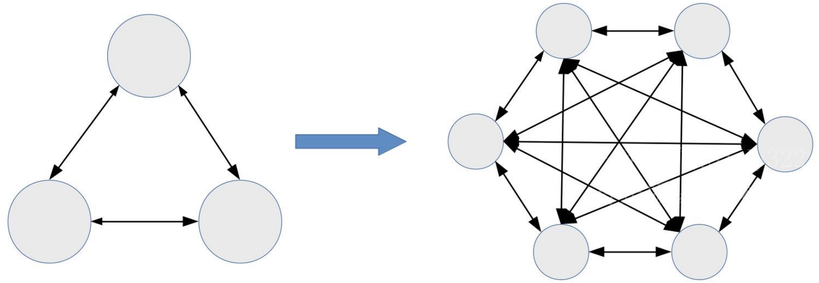
目录

1. 微服务架构6大陷阱

微服务陷阱



1. 拆分粒度太细，服务关系复杂

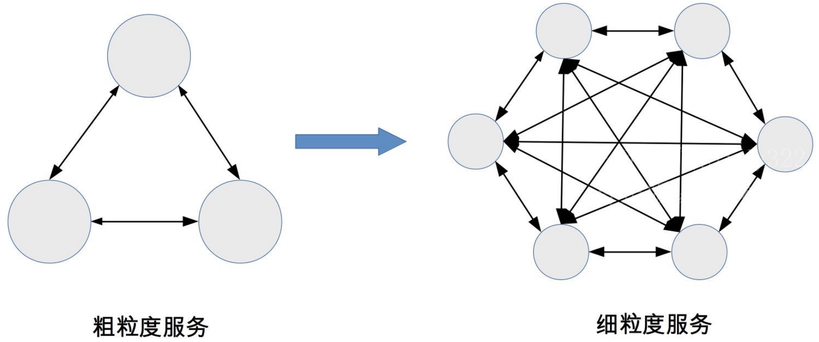


需求分析、方案设计、测试、部署……难度都会增加。

例如：
1. 分布式服务如何保证数据一致性；
2. 分析设计的时候需要考虑的影响点变多。

拆的太细，降低了什么复杂度，提升了什么复杂度？

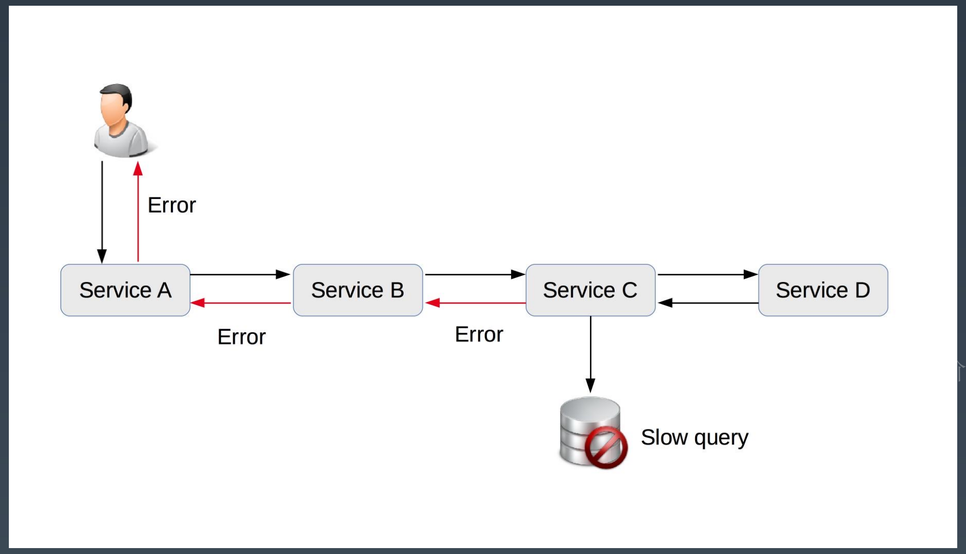
2. 拆分粒度太细，团队效率下降

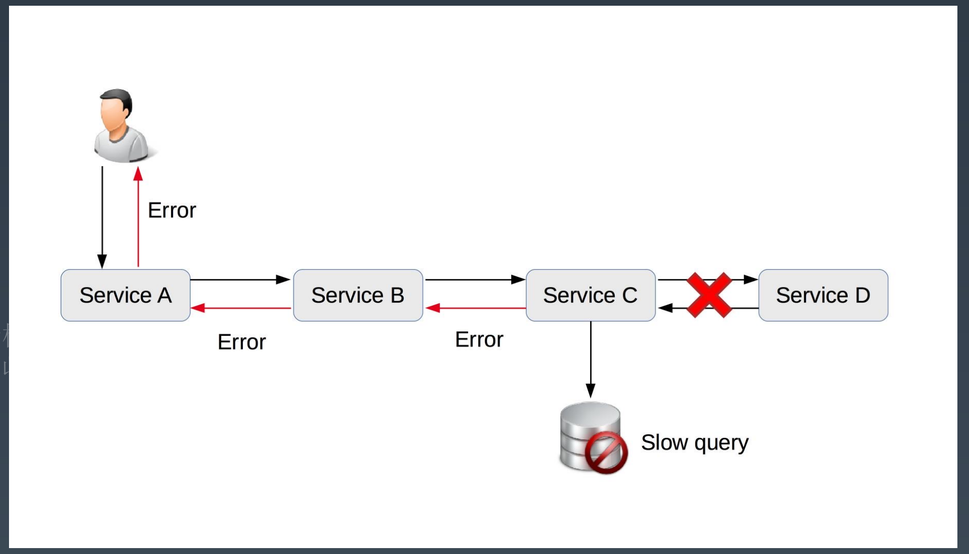


需求分析、方案设计、测试、部署……工作量都会增加。

例如：
1. 接口设计数量，1次请求由两个服务处理，接口数量1个，5个服务处理，接口数量4个，接口设计、接口联调、接口测试等工作量都会大大增加；
2. 某个业务功能上线，需要升级的系统数量会增加。

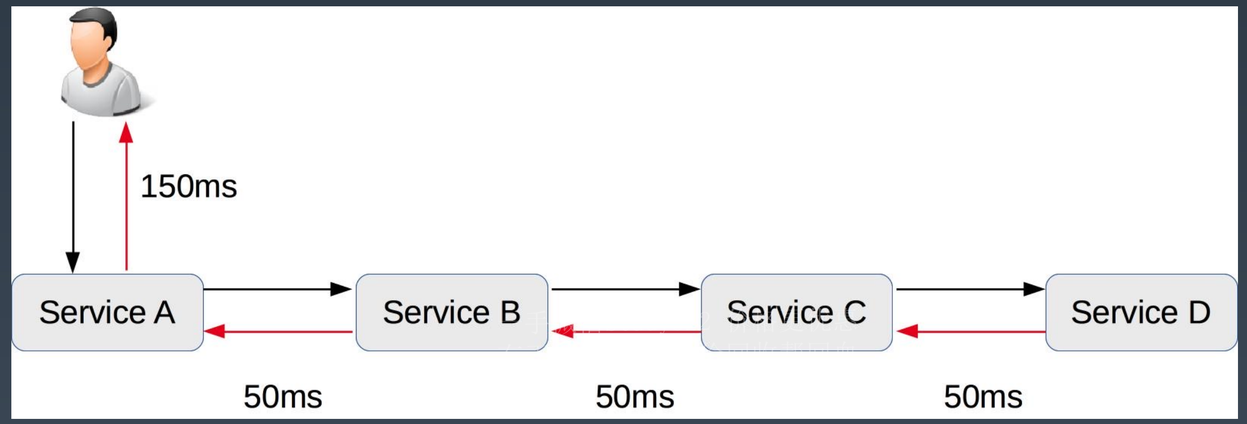
3. 拆分粒度太细，问题定位困难





故障扩散：单个微服务的故障，会导致多个微服务异常，监控系统一片红，到处都在告警，但是不知道根因。

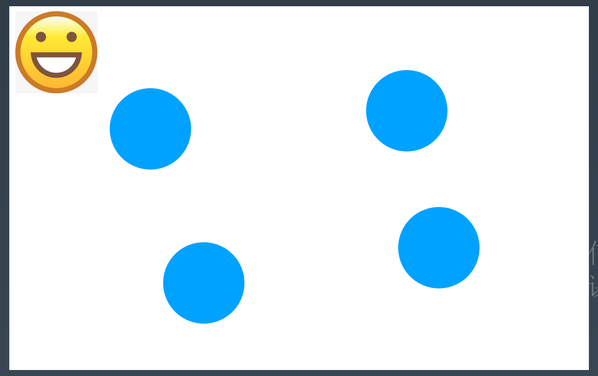
4. 拆分粒度太细，系统性能下降

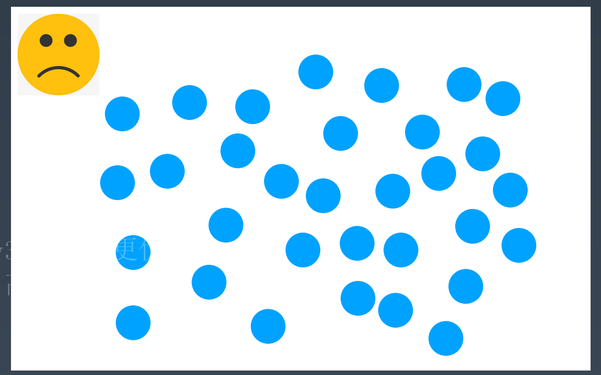


调用链越长，单次请求耗时会更长。

拆分后单个服务的处理性能会提升，是否可以弥补调用链带来的消耗？

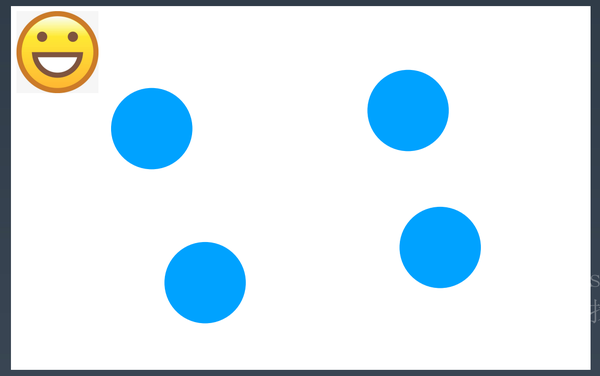
5. 缺乏基础设施，无法快速交付

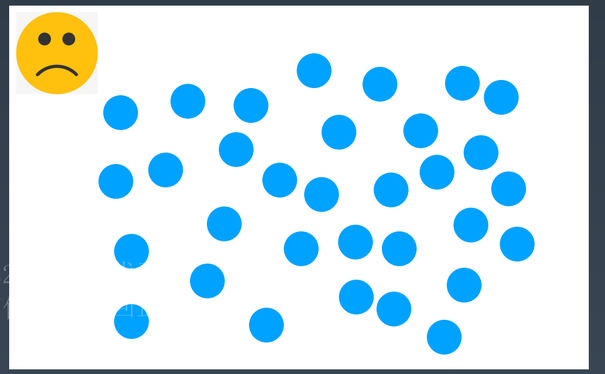




1. 没有自动化测试支撑，每次测试时需要测试大量接口；
2. 没有自动化部署支撑，人肉运维，手都敲麻了；
3. 没有自动化监控，每次故障定位都需要人工查几十台机器几百个微服务的各种状态和各种日志文件。

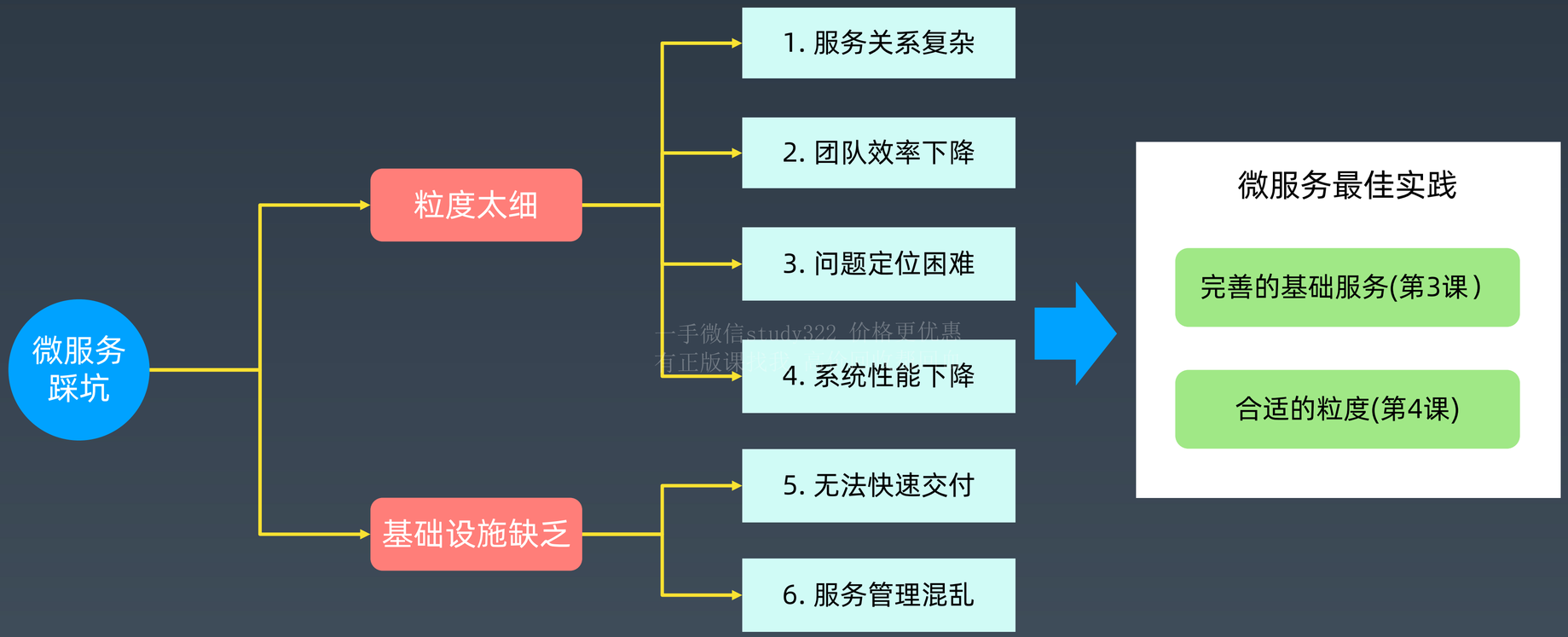
6. 缺乏基础设施，服务管理混乱





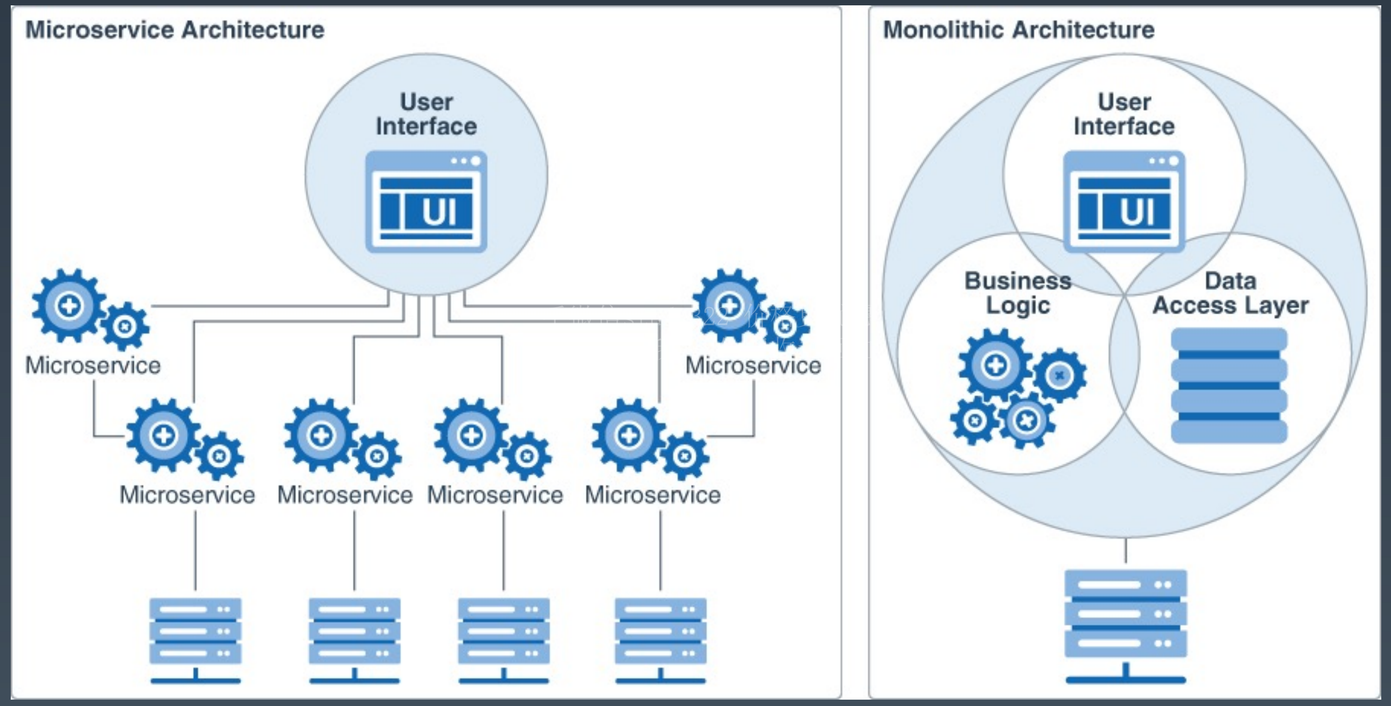
1. 服务路由：假设某个微服务有60个节点，部署在20台机器上，那么其他依赖的微服务如何知道这个部署情况呢？
2. 服务故障隔离：假设上述例子中的60个节点有5个节点发生故障了，依赖的微服务如何处理这种情况呢？
3. 服务注册和发现：同样是上述的例子，现在我们决定从60个节点扩容到80个节点，或者将60个节点缩减为40个节点，新增或者减少的节点如何让依赖的服务知道呢？

微服务陷阱 - 如何应对？



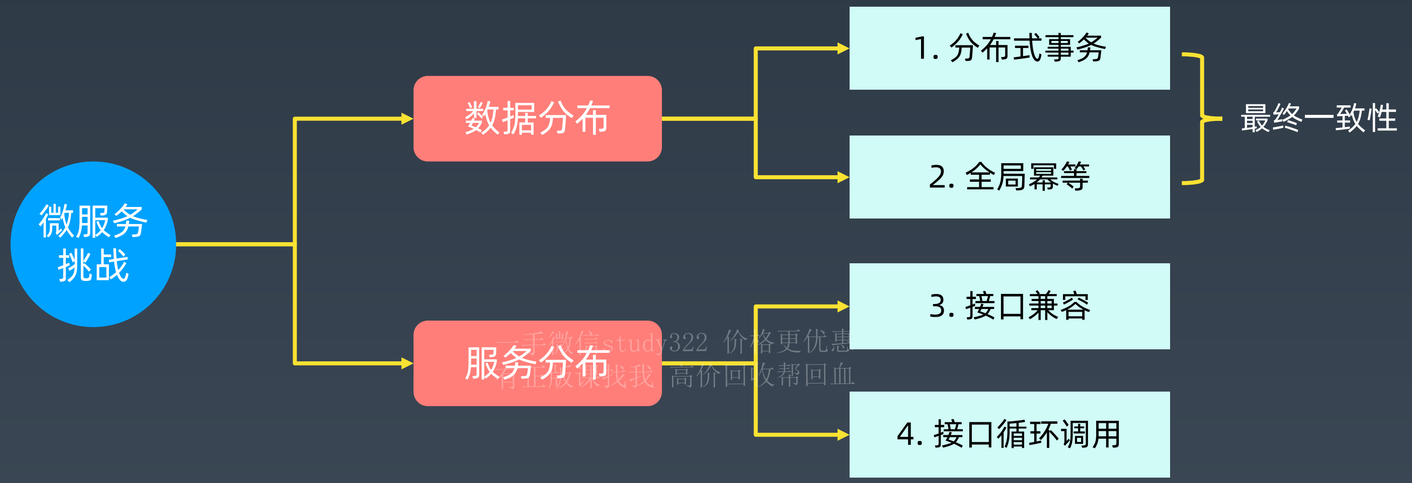
2. 微服务架构4大挑战

微服务架构示意



参考链接

微服务技术挑战



只拆分代码，不拆分数据库，算不算微服务？

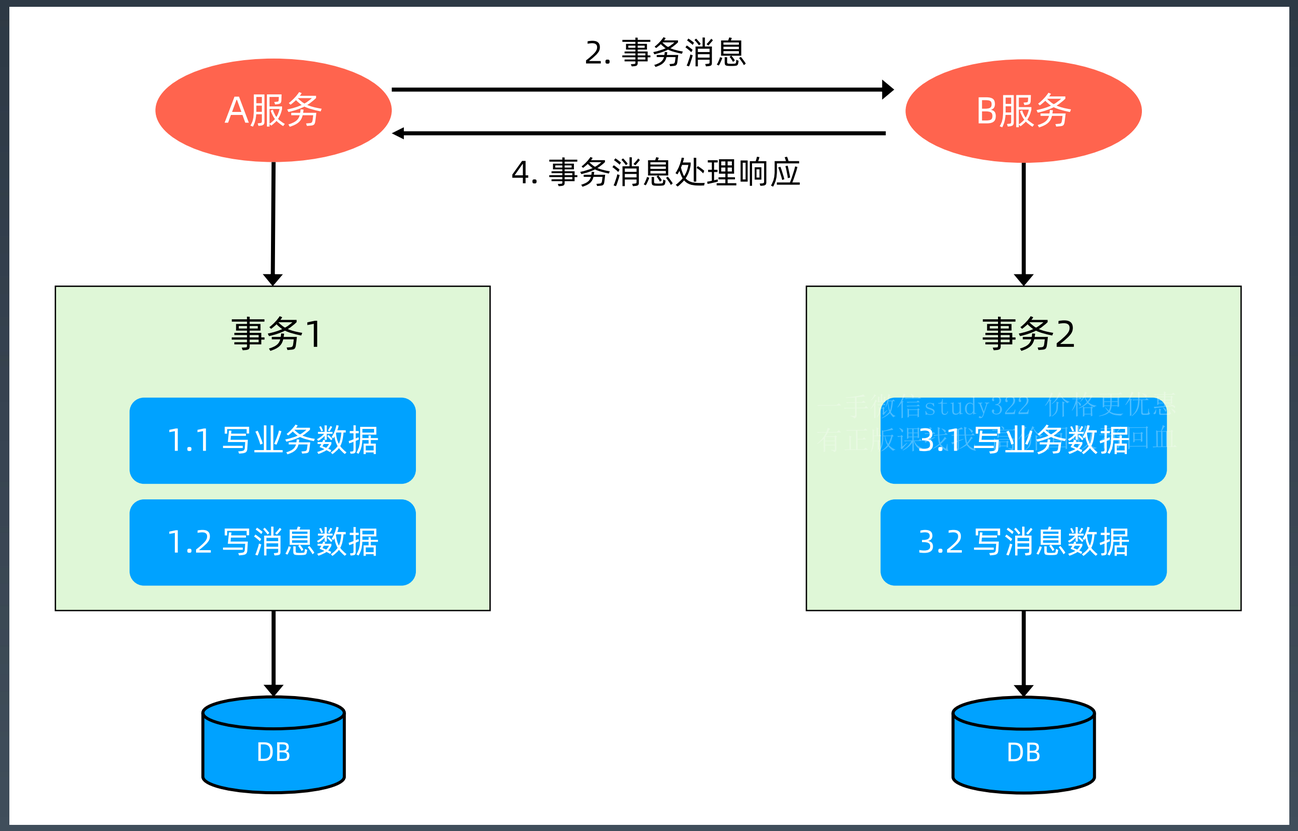
BASE 理论之最终一致性

【定义】
BASE：Basically Available（基本可用）、Soft state（软状态）和 Eventually consistent（最终一致性）。

【核心思想】
分布式系统由于 CAP 理论限制，无法做到强一致性，但每个应用都可以根据自身业务特点，采用适当的方式来使系统达到最终一致性。

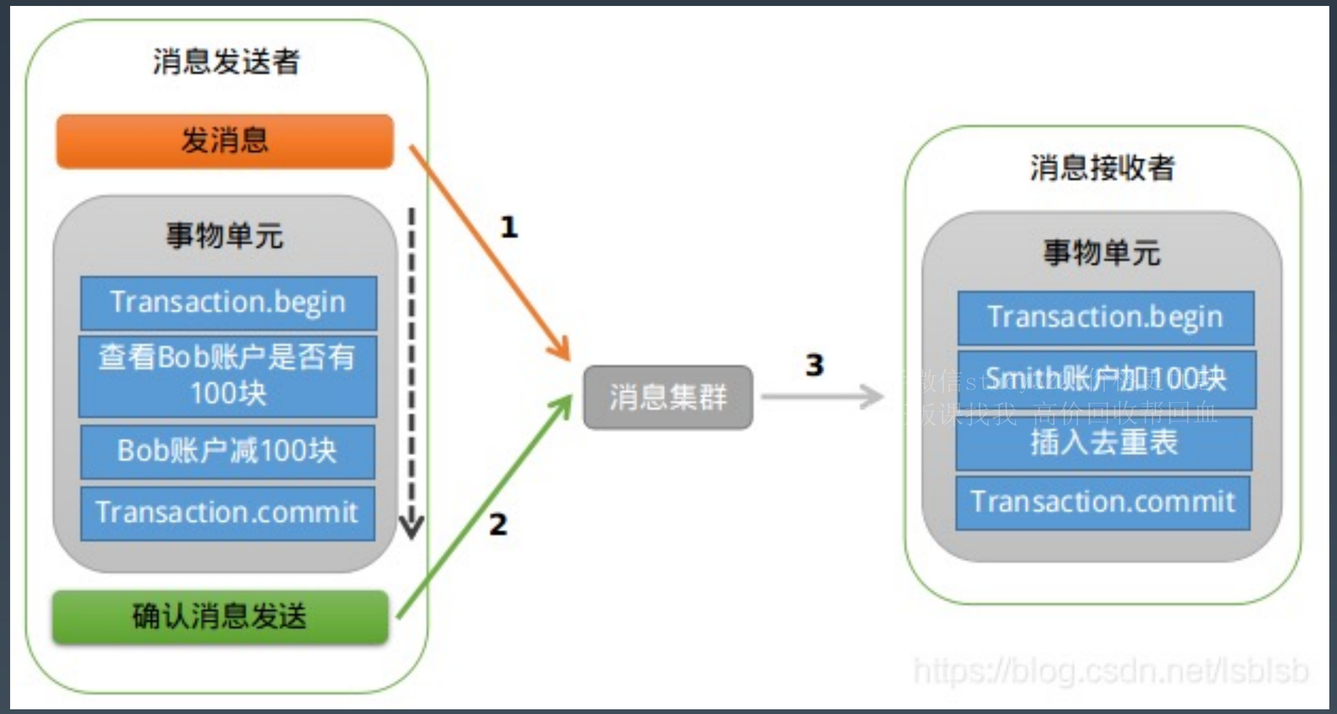
【设计关键】
1. 适当的方式：有哪些方式？分布式系统只有一种方式：消息，区别只是传递消息的方式和消息内容不同，例如消息队列、接口调用、ZooKeeper 事件。
2. 最终一致性：什么叫“最终”？其实就是从不一致到一致的持续时间延迟，上图中 t2-t1。

业务级分布式事务 - 本地事务消息



1. 如果“2. 事务消息”丢失，A服务会不断重试；
2. 如果“4. 处理响应”丢失，A服务会不断重试；
3. B 服务重复收到2. 事务消息，然后检查消息表是否已经处理过，已经处理过就直接返回处理结果，没有处理就正常处理(幂等)。

业务级分布式事务 - 消息队列事务消息

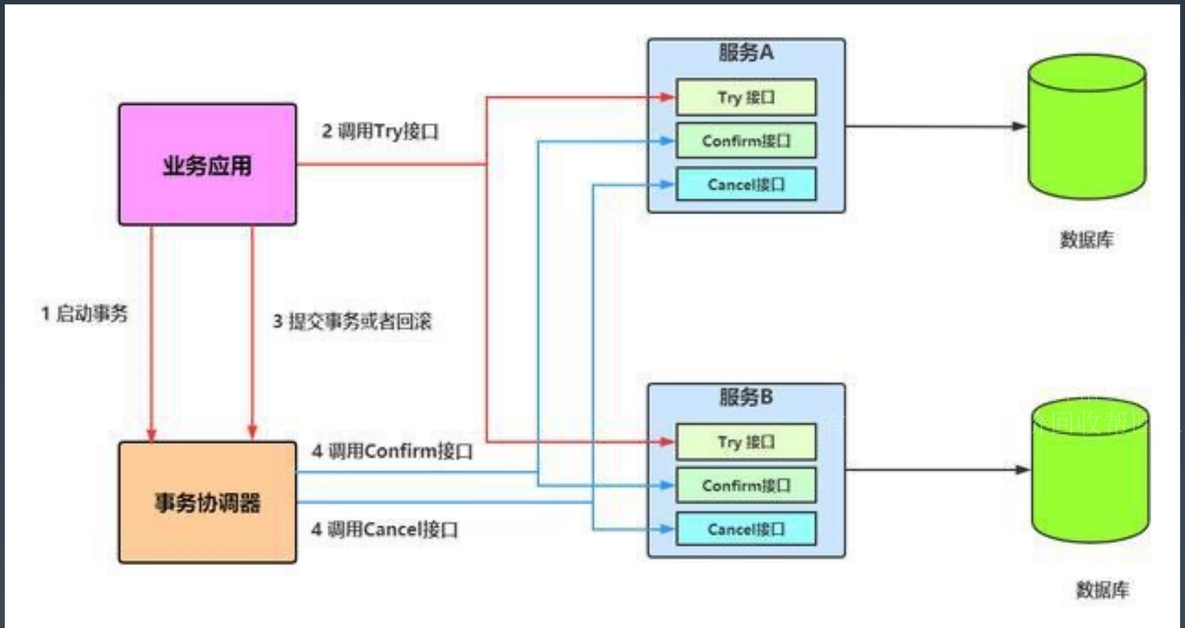


【RocketMQ】
1. 发送 Prepared 消息时，会拿到消息的地址；
2. 执行本地事务；
3. 通过1拿到的地址去修改消息状态；
4. RocketMQ 会定期扫描消息集群中的事务消息，如果发现了 Prepared 消息，它会向消息发送者确认，RocketMQ 会根据发送端设置的策略来决定是回滚还是继续发送确认消息。

学习链接

本地事务消息和消息队列事务消息两个方案，你觉得哪个更好？

业务级分布式事务 - TCC



【Try】
1. 服务A修改订单为“Paying”状态；
2. 服务B冻结库存2，剩余可用库存98。

【Confirm】
3. 服务A修改订单为“Paid”；
4. 服务B冻结库存为0，剩余可用库存98。

【Cancel】
3.1 服务A修改订单为“Canceled”；
4.1 服务B冻结库存为0，剩余可用库存100。

学习链接

应用层面的 2PC，事务协调器就是业务代码，Confirm 接口和 Cancel 接口要实现幂等（全局 ID + 状态）。

全局幂等

【幂等技术本质】
分布式数据只能通过消息来实现最终一致性，而消息可能会丢失，因此需要重试，重试就需要保证幂等。

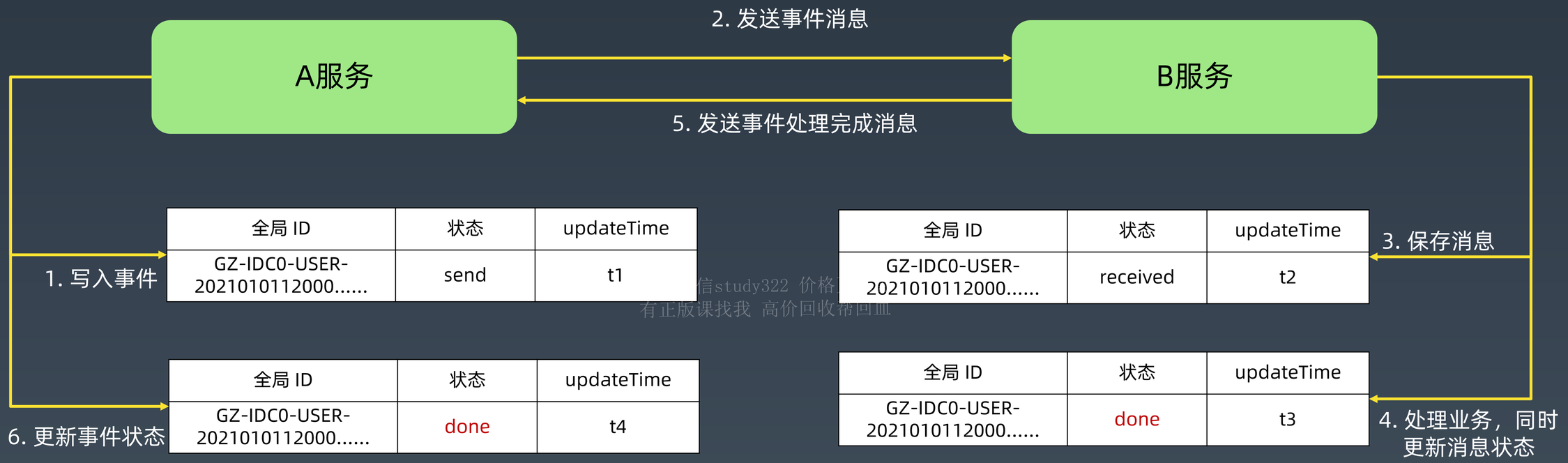
【幂等定义】
幂等（idempotent、idempotence）是一个数学与计算机学概念，常见于抽象代数中。 在编程中一个幂等操作的特点是其任意多次执行所产生的影响均与一次执行的影响相同。

【全局幂等】
全局范围内的幂等，保证每个幂等操作都是全局唯一的。

【设计关键】
1. 全局唯一 ID；
2. 状态机。

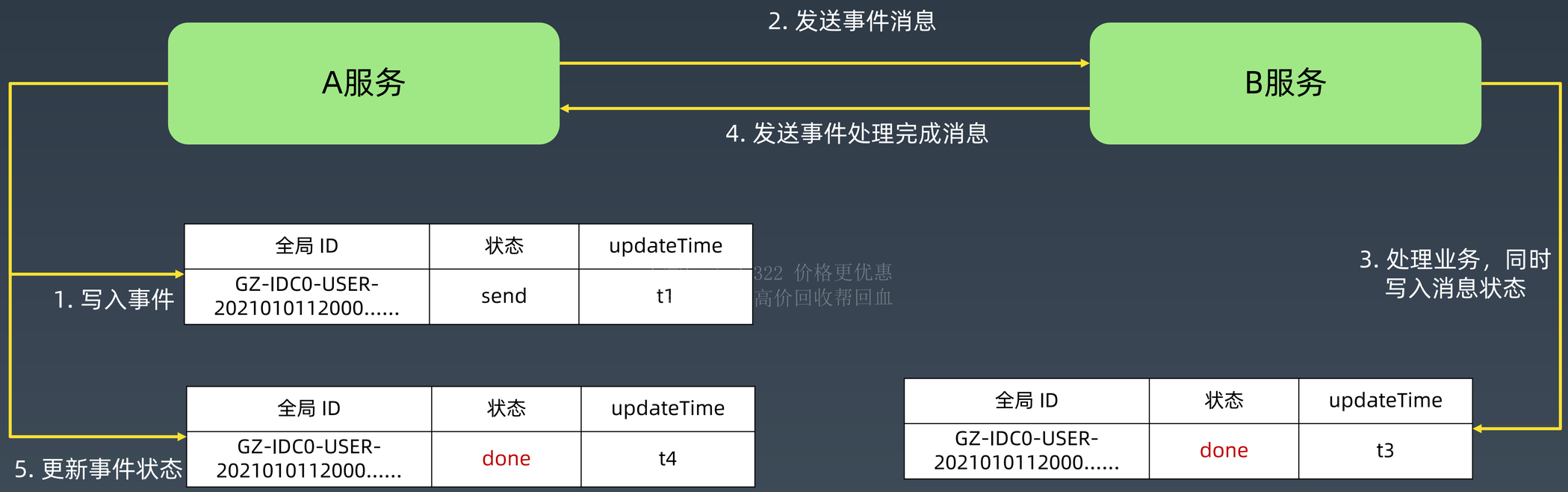
Kafka、RocketMQ 都是高可用的，为何事件消息还会丢？

全局幂等设计示例 - 正常处理1

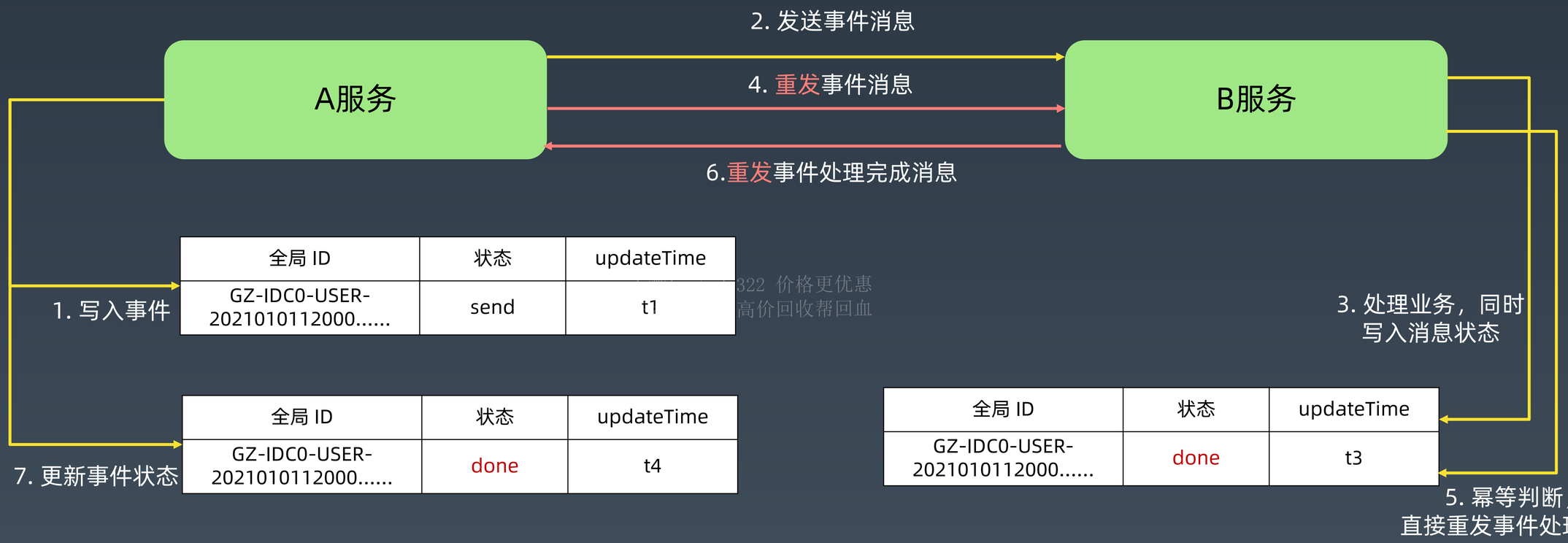


第4步如果处理业务成功后，更新消息状态失败，如何应对这种情况？

全局幂等设计示例 - 正常处理2



全局幂等设计示例 - 幂等处理



接口兼容 和 接口循环调用

极客时间 训练营

接口兼容

【问题】
某个微服务的某些接口升级，依赖这些接口的微服务不一定能够全部同时升级。

【解决方案】
1. 接口多版本，直接拷贝一份旧接口代码，在旧接口代码上修改，接口 URL 加上 v1/v2 这种标识；
2. 接口逻辑兼容，同一份接口代码，兼容新旧逻辑，容易互相影响，且旧接口下线时又要修改代码（不推荐）。

接口循环调用

【问题】
某次业务处理过程中，A调用B，B又过来调用A，A的处理又进入了之前的处理逻辑，导致循环调用，整个业务进入死循环。

【解决方案】
几乎没有好的解决方案，运气好靠测试发现，运气不好靠上线发现。

【举例】
1. 用户登录服务调用风控服务进行安全检查；
2. 风控服务又来调用登录服务获取用户登录地址信息；
3. 获取登录地信息的接口又依赖了风控服务进行安全检查。

不是说要遵循 DRY 原则么？为何还要拷贝代码？

本节思维导图


微服务架构深入剖析
6大陷阱
拆分粒度太细
服务关系复杂
团队效率下降
问题定位困难
系统性能下降
基础设施不完善
无法快速交付
服务管理混乱
4大挑战
1. 分布式事务
本地消息表
消息队列事务消息
TCC
2. 全局幂等
全局唯一ID
状态机
3. 接口兼容
多版本接口
4. 接口循环调用
自求多福

随堂测验

极客时间 训练营

【判断题】

1. 微服务拆分降低了系统整体的复杂度。
2. 如果没有基础设施支撑，微服务会增加团队工作量。
3. 最终一致性意味着数据最终达到一致性状态就可以了，不需要去考虑数据不一致的持续时间。
4. 全局幂等一定要设计事件的全局唯一 ID。
5. 遵循 DRY 原则，接口兼容的时候不要拷贝代码。

【思考题】

回忆一下你在工作中遇到过课程中涉及的哪些陷阱和技术挑战，思考一下主要的原因是什么？

Q&A

茶歇时间

八卦，趣闻，内幕……

THANKS