超级教程系列

微服务架构的分布式事务解决方案

分布式架构系统中，分布式事务是一个绕不过去的挑战！
微服务架构的流行，让分布式事务问题日益突出！

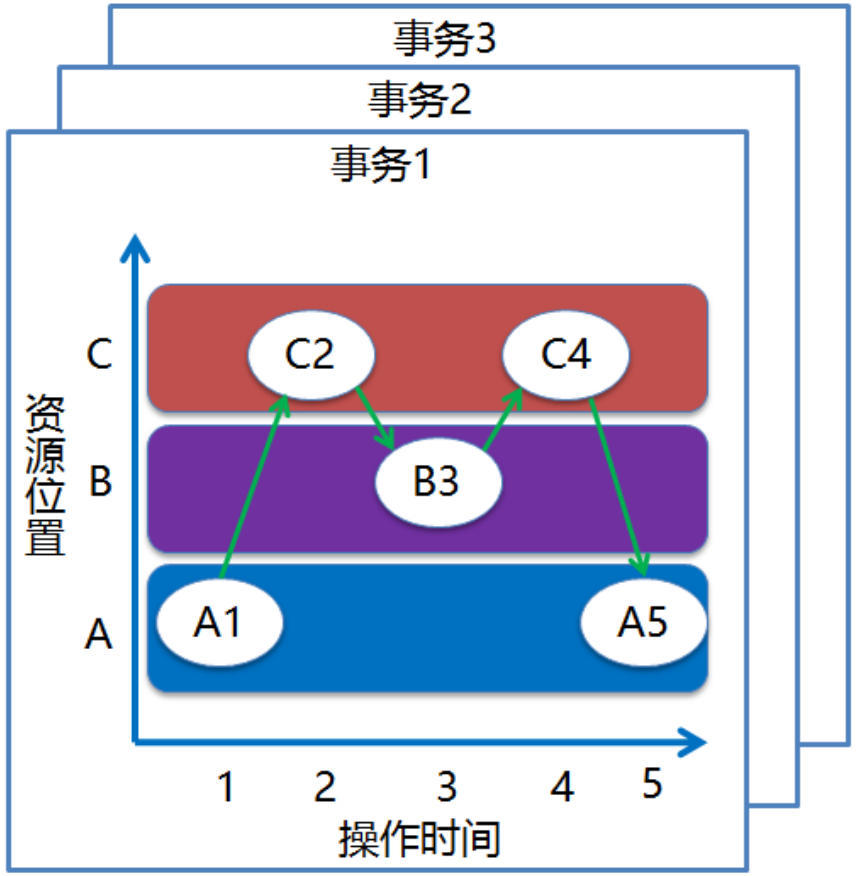
龙果学院 roncoo.com
讲师：吴水成（水到渠成）
邮箱：840765167@qq.com

超级教程系列
《微服务架构的分布式事务解决方案》

第03节--常用的分布式事务解决方案介绍

龙果学院 roncoo.com
讲师：吴水成（水到渠成）
邮箱：840765167@qq.com

事务



事务:

- 由一组操作构成的**可靠**、**独立**的工作单元

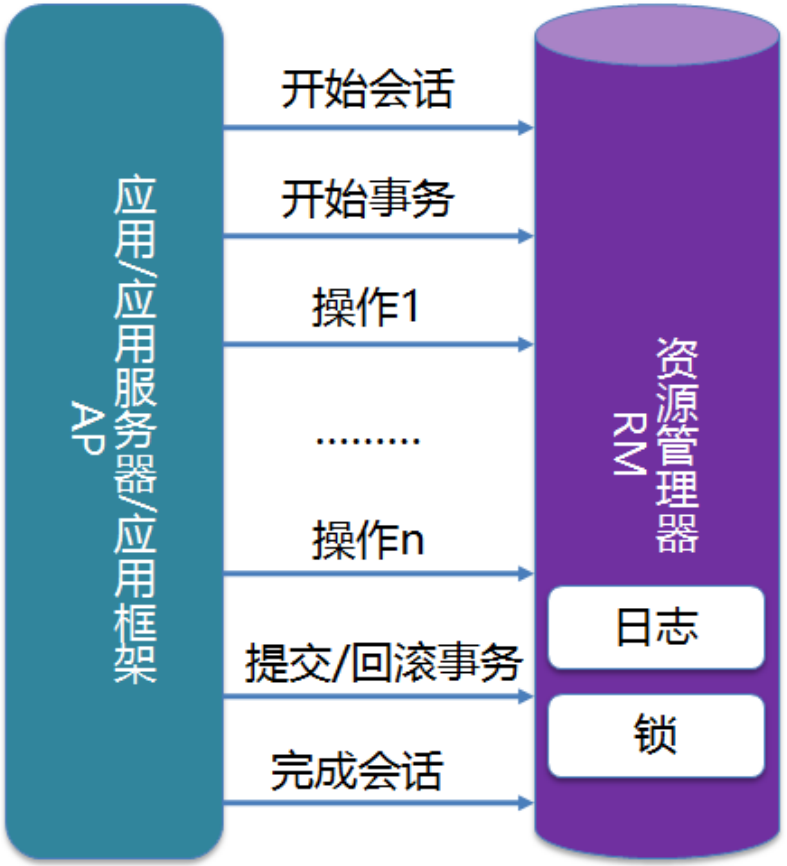
ACID:

- Atomicity(原子性)
- Consistency(一致性)
- Isolation(隔离性)
- Durability(持久性)

难点:

- 高度并发
- 资源分布
- 大时间跨度

本地事务



本地事务

- 事务由资源管理器（如DBMS）**本地**管理

优点

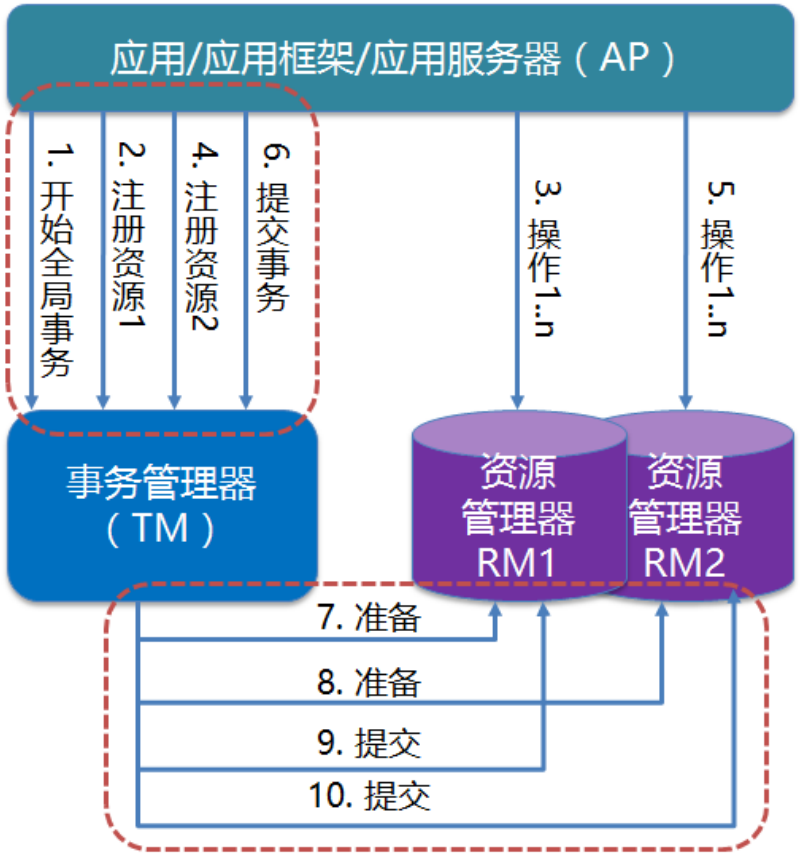
- 支持严格的ACID属性
- 可靠
- 高效
- 状态可以只在资源管理器中维护
- 应用编程模型简单(在框架或平台的支持)

局限

- 不具备分布事务处理能力
- 隔离的最小单位由资源管理器决定，如数据库中的一条记录

- 在单个数据库的本地并且限制在单个进程内的事务
- 本地事务不涉及多个数据来源

全局事务（DTP模型）--标准分布式事务



全局事务

- 事务由全局事务管理器全局管理

事务管理器

- 管理全局事务状态与参与的资源，协同资源的一致提交/回滚

TX协议

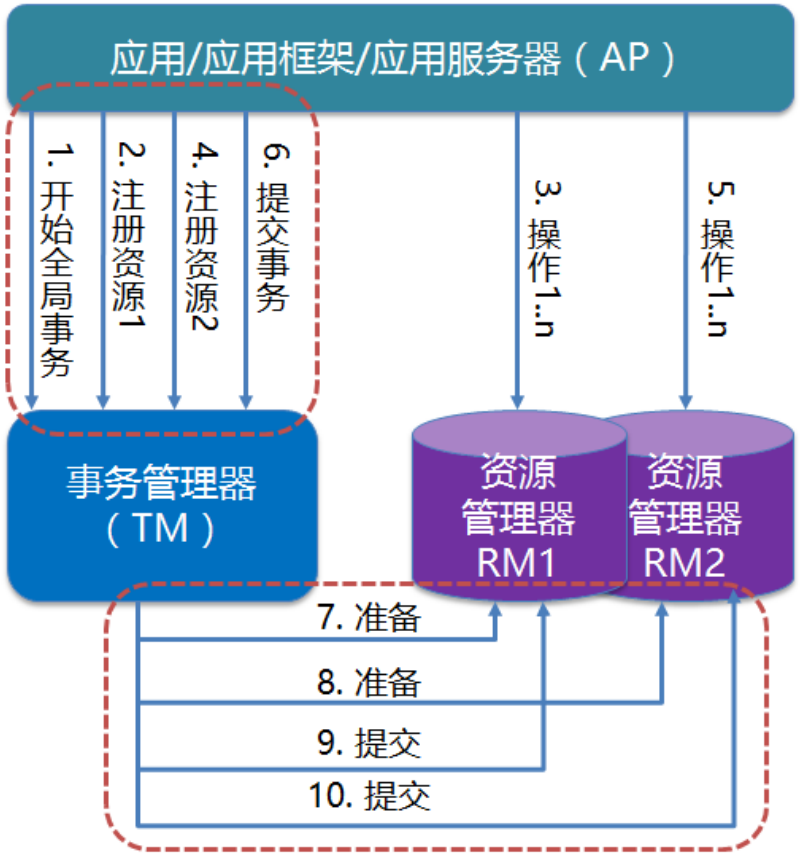
- 应用或应用服务器与事务管理器的接口

XA协议

- 全局事务管理器与资源管理器的接口

- AP(Application Program)：也就是应用程序，可以理解为使用 DTP 的程序；
- RM(Resource Manager)：资源管理器（这里可以是一个 DBMS，或者消息服务器管理系统）应用程序通过资源管理器对资源进行控制，资源必须实现 XA 定义的接口；
- TM(Transaction Manager)：事务管理器，负责协调和管理事务，提供给 AP 应用程序编程接口以及管理资源管理器。
- 事务管理器控制着全局事务，管理事务生命周期，并协调资源。资源管理器负责控制和管理实际资源。

全局事务（DTP模型）--XA



全局事务

- 事务由全局事务管理器全局管理

事务管理器

- 管理全局事务状态与参与的资源，协同资源的一致提交/回滚

TX协议

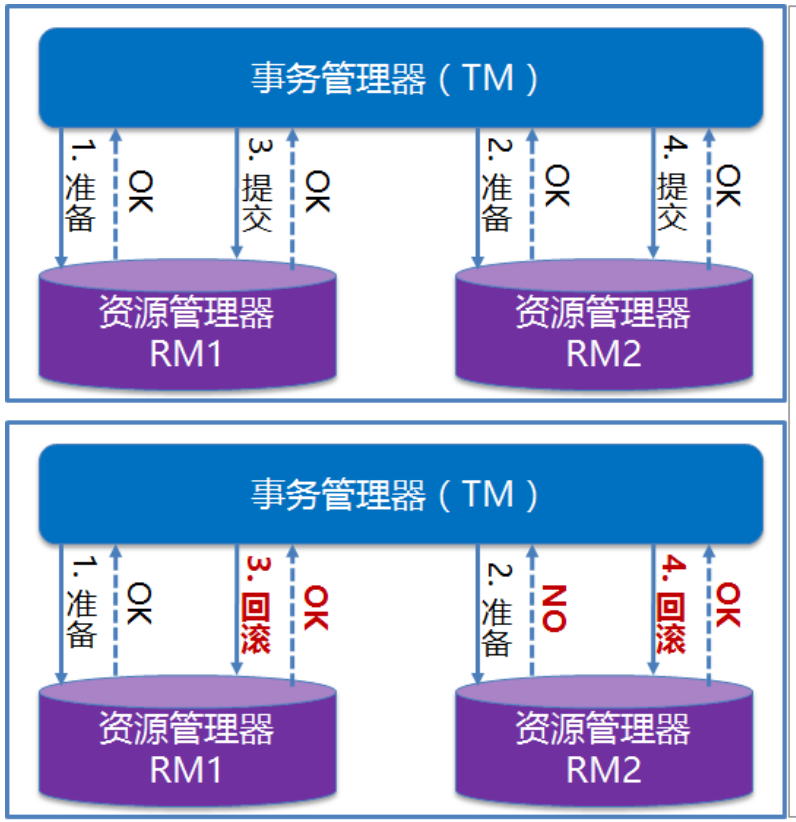
- 应用或应用服务器与事务管理器的接口

XA协议

- 全局事务管理器与资源管理器的接口

- XA是由X/Open组织提出的分布式事务的规范。XA规范主要定义了(全局)事务管理器(TM)和(局部)资源管理器(RM)之间的接口。主流的关系型数据库产品都是实现了XA接口的。
- XA接口是双向的系统接口，在事务管理器（TM）以及一个或多个资源管理器（RM）之间形成通信桥梁。
- XA之所以需要引入事务管理器是因为，在分布式系统中，从理论上讲两台机器理论上无法达到一致的状态，需要引入一个单点进行协调。
- 由全局事务管理器管理和协调的事务，可以跨越多个资源（如数据库或JMS队列）和进程。全局事务管理器一般使用 XA 二阶段提交协议与数据库进行交互。

两阶段提交(Two Phase Commit)



准备操作与ACID

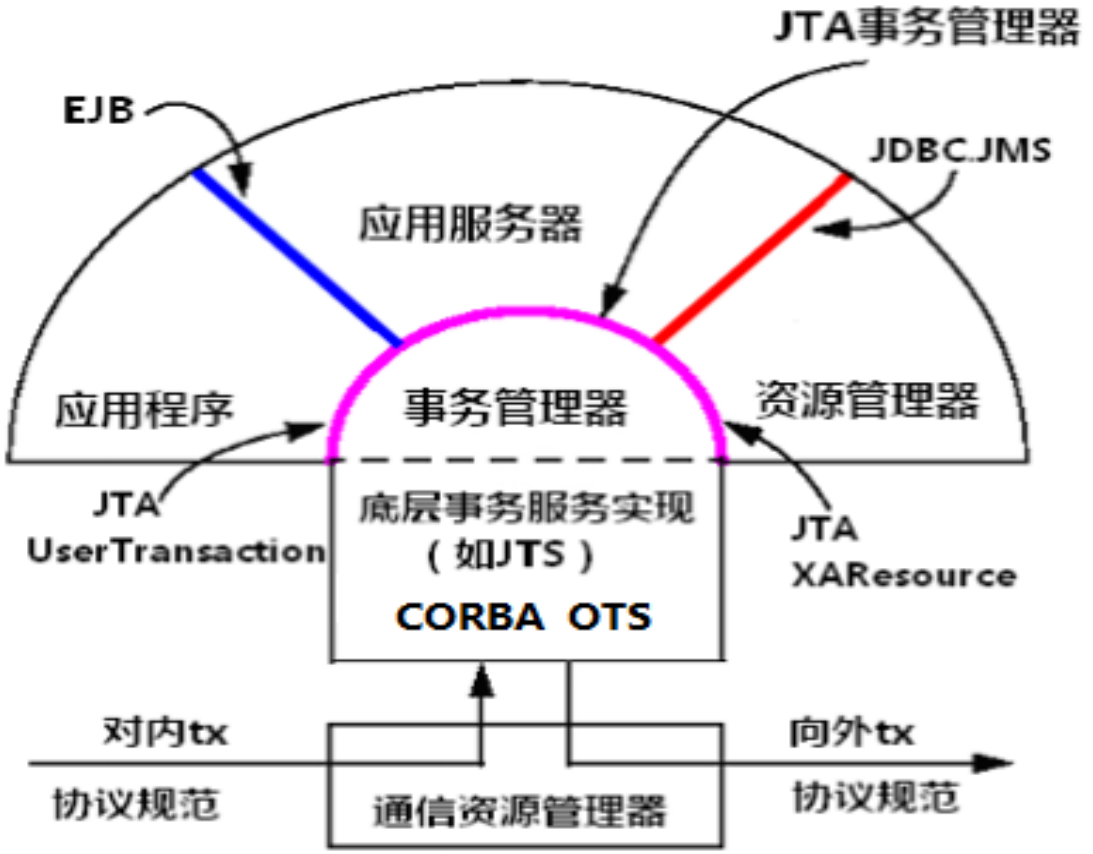
- A: 准备后，仍可提交与回滚
- C: 准备时，一致性检查必须OK
- I: 准备后，事务结果仍然只在事务内可见
- D: 准备后，事务结果已经持久

局限

- 协议成本 (准备操作是一定必须的吗）
- 准备阶段的持久成本
- 全局事务状态的持久成本
- 潜在故障点多带来的脆弱性
- 准备后，提交前的故障引发一系列隔离与恢复难题

- 两阶段提交协议（Two-phase commit protocol）是XA用于在全局事务中协调多个资源的机制。
- TM和RM间采取两阶段提交(Two Phase Commit)的方案来解决一致性问题。
- 两阶段提交需要一个协调者（TM）来掌控所有参与者节点（RM）的操作结果并且指引这些节点是否需要最终提交。

JavaEE平台中的分布式事务实现



JavaEE分布式事务服务层次示意图，图中的粉红小半圆代表JTA规范

- JTA（Java Transaction API）：面向应用、应用服务器与资源管理器的高层事务接口。
- JTS（Java Transaction Service）：JTA事务管理器的实现标准，向上支持JTA，向下通过CORBA OTS实现跨事务域的互操作性。
- EJB：基于组件的应用编程模型，通过声明式事务管理进一步简化事务应用的编程。

- 优点
  - 简单一致的编程模型
  - 跨域分布处理的ACID保证

- 局限
  - DTP模型本身的局限

标准分布式事务解决方案的利弊

- 优点：严格的ACID

- 缺点：效率非常低（微服务架构下已不太适用）

- 全局事务方式下，全局事务管理器（TM）通过XA接口使用二阶段提交协议（ 2PC ）与资源层（如数据库）进行交互。使用全局事务，数据被Lock的时间跨整个事务，直到全局事务结束。

- 2PC 是反可伸缩模式，在事务处理过程中，参与者需要一直持有资源直到整个分布式事务结束。这样，当业务规模越来越大的情况下，2PC 的局限性就越来越明显，系统可伸缩性会变得很差。

- 与本地事务相比，XA 协议的系统开销相当大，因而应当慎重考虑是否确实需要分布式事务。而且只有支持 XA 协议的资源才能参与分布式事务。

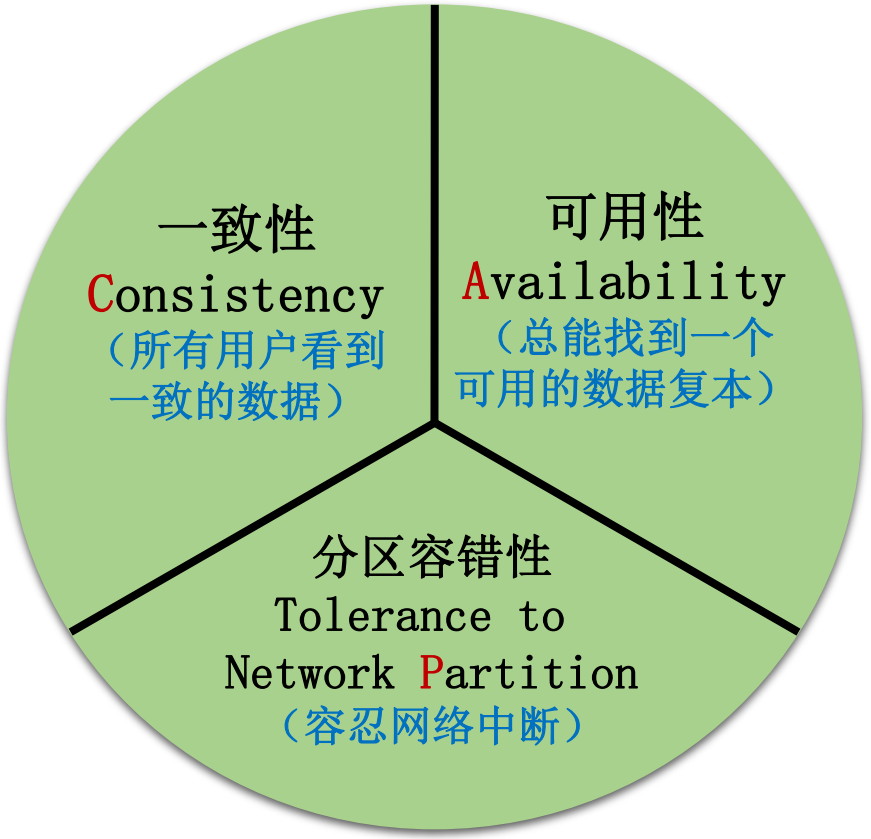
BASE理论

- BASE
- BA: Basic Availability 基本业务可用性（支持分区失败）
- S: Soft state 柔性状态（状态允许有短时间不同步，异步）
- E: Eventual consistency 最终一致性（最终数据是一致的，但不是实时一致）
- 原子性（A）与持久性（D）必须根本保障
- 为了可用性、性能与降级服务的需要，只有降低一致性(C)与 隔离性(I)的要求

- 酸碱平衡(ACID-BASE Balance)

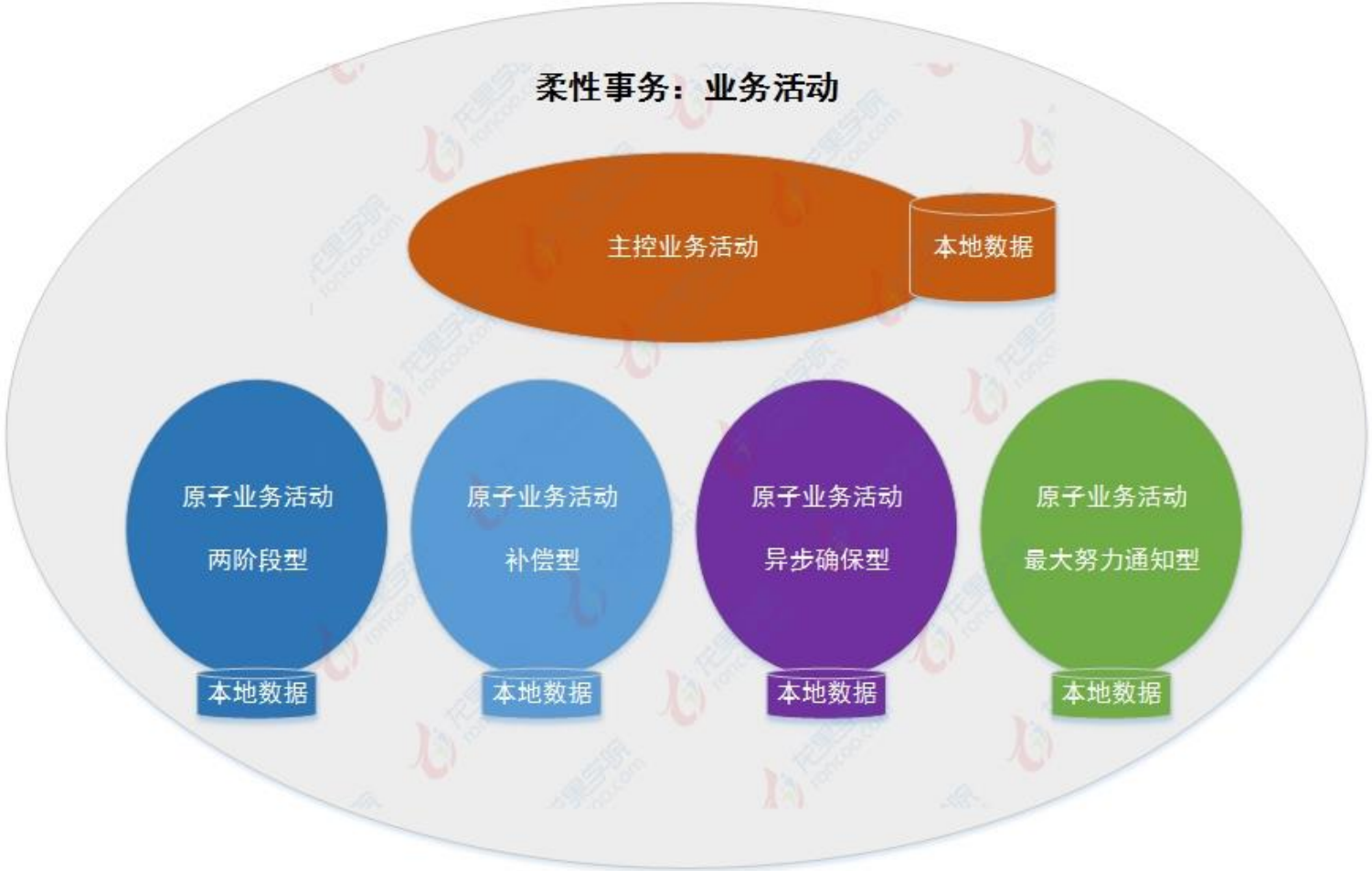
CAP定理

- 定理: 对于共享数据系统，最多只能同时拥有CAP其中的两个，没法三者兼顾。
  - 任两者的组合都有其适用场景
  - 真实系统应当是ACID与BASE的混合体
  - 不同类型的业务可以也应当区别对待



- 结论：分布式系统中，最重要的是满足业务需求，而不是追求抽象、绝对的系统特性。

柔性事务



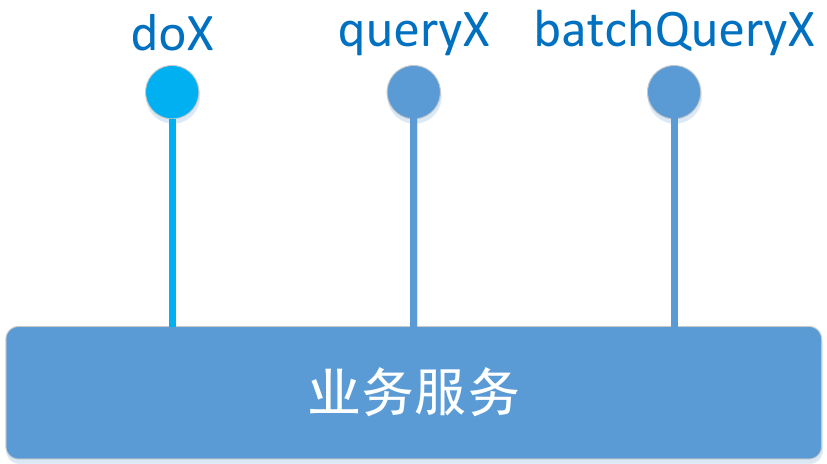
- 两阶段型
- 补偿型
- 异步确保型
- 最大努力通知型

柔性事务中的服务模式

- 可查询操作
- 幂等操作
- TCC操作
- 可补偿操作

注：服务模式是柔性事务流程中的特殊操作实现（实现上对应业务服务要提供相应模式的功能接口），
还不算是某一种柔性事务解决方案。

柔性事务中的服务模式：可查询操作



- **服务操作的可标识性**
  - 服务操作具有全局唯一标识
    - 可以使用业务单据号（如订单号）
    - 或者使用系统分配的操作流水号（如支付记录流水号）
    - 或者使用操作资源的组合组合标识（如商户号+商户订单号）
  - 操作有唯一的、确定的时间(约定以谁的时间为准)
- **单笔查询**
  - 使用全局唯一的服务操作标识，查询操作执行结果
  - 注意状态判断，小心“处理中”的状态
- **批量查询**
  - 使用时间区段与(或)一组服务操作的标识，查询一批操作执行结果

柔性事务中的服务模式：幂等操作

- **幂等性（Idempotenty）**

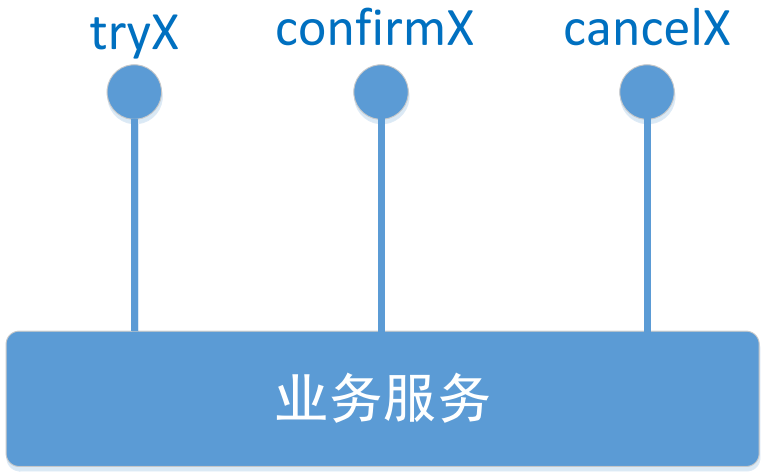
  f(f(x)) = f(x)

- **幂等操作**
  - 重复调用多次产生的业务结果与调用一次产生的业务结果相同

- **实现方式一**
  - 通过业务操作本身实现幂等性

- **实现方式二**
  - 系统缓存所有请求与处理结果
  - 检测到重复请求之后，自动返回之前的处理结果

柔性事务中的服务模式：TCC操作



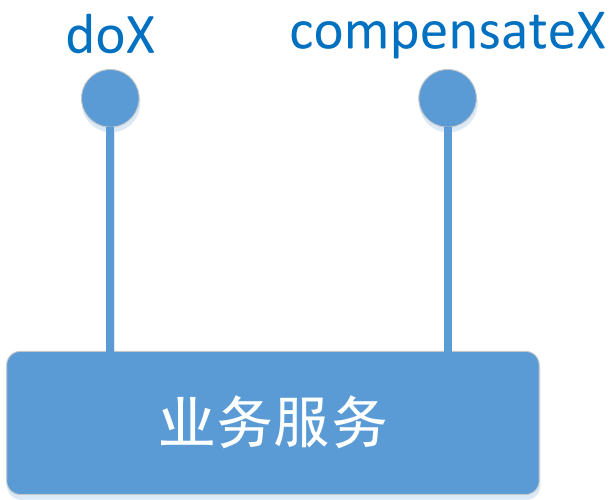
- **Try: 尝试执行业务**
  - 完成所有业务检查(一致性)
  - 预留必须业务资源(准隔离性)
- **Confirm:确认执行业务**
  - 真正执行业务
  - 不作任何业务检查
  - 只使用Try阶段预留的业务资源
  - Confirm操作要满足幂等性
- **Cancel: 取消执行业务**
  - 释放Try阶段预留的业务资源
  - Cancel操作要满足幂等性

- **与2PC协议比较**
  - 位于业务服务层而非资源层
  - 没有单独的准备(Prepare)阶段，Try操作兼备资源操作与准备能力
  - Try操作可以灵活选择业务资源的锁定粒度(以业务定粒度)
  - 较高开发成本

误区：很多人把两阶段型操作等同于两阶段提交协议2PC操作。

其实TCC操作也属于两阶段型操作。

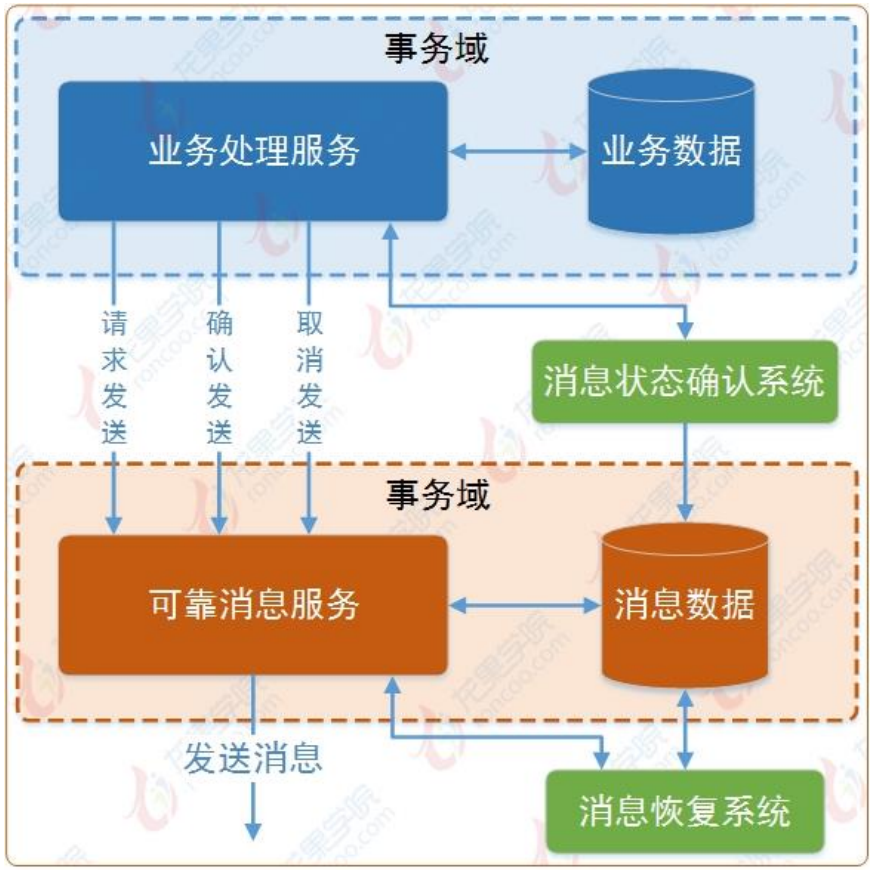
柔性事务中的服务模式：可补偿操作



- **do: 真正执行业务**
  - 完成业务处理
  - 业务执行结果外部可见
- **compensate:业务补偿**
  - 抵销(或部分抵销)正向业务操作的业务结果
  - 补偿操作满足幂等性
- **约束**
  - 补偿在业务上可行
  - 由于业务执行结果未隔离、或者补偿不完整带来的风险与成本可控

（TCC操作中的Confirm操作和Cancel操作，其实也可以看作是补偿操作）

柔性事务解决方案：可靠消息最终一致（异步确保型）



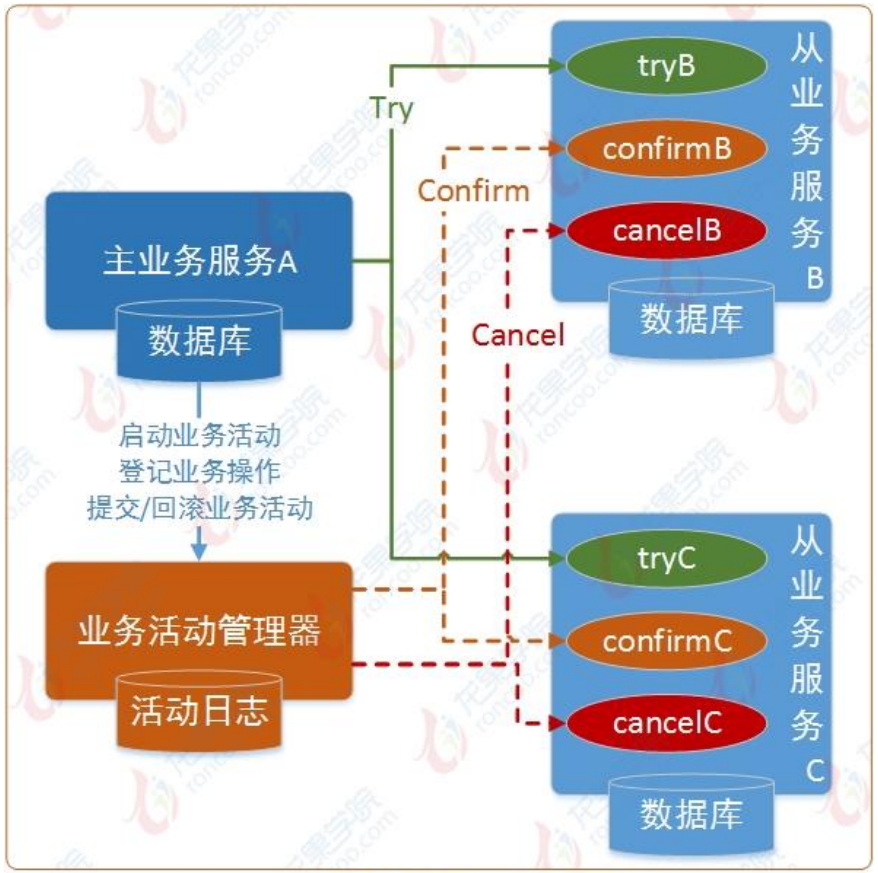
可靠消息最终一致（异步确保型）

- **实现**
- 业务处理服务在业务事务提交前，向实时消息服务请求发送消息，实时消息服务只记录消息数据，而不真正发送。业务处理服务在业务事务提交后，向实时消息服务确认发送。只有在得到确认发送指令后，实时消息服务才真正发送
- **消息**
- 业务处理服务在业务事务回滚后，向实时消息服务取消发送。消息状态确认系统定期找到未确认发送或回滚发送的消息，向业务处理服务询问消息状态，业务处理服务根据消息ID或消息内容确定该消息是否有效
- **约束**
- 被动方的处理结果不影响主动方的处理结果， 被动方的消息处理操作是幂等操作
- **成本**
- 可靠消息系统建设成本
- 一次消息发送需要两次请求，业务处理服务需实现消息状态回查接口
- **优点、适用范围**
- 消息数据独立存储、独立伸缩，降低业务系统与消息系统间的耦合
- 对最终一致性时间敏感度较高，降低业务被动方实现成本

柔性事务解决方案：可靠消息最终一致（异步确保型）

- **用到的服务模式**
  - 可查询操作、幂等操作
- **方案特点**
  - 兼容所有实现JMS标准的MQ中间件（本教程中实现的方案）
  - 确保业务数据可靠的前提下，实现业务数据的最终一致（理想状态下基本是准实时一致）
- **行业应用案例**
  - 支付宝、eBay（BASE）、去哪儿……

柔性事务解决方案：TCC（两阶段型、补偿型）



TCC（两阶段型、补偿型）

- **实现**
- 一个完整的业务活动由一个主业务服务与若干从业务服务组成
- 主业务服务负责发起并完成整个业务活动
- 从业务服务提供TCC型业务操作
- 业务活动管理器控制业务活动的一致性，它登记业务活动中的操作， 并在业务活动提交时确认所有的TCC型操作的confirm操作，在业务活动取消时调用所有TCC型操作的cancel操作
- **成本**
- 实现TCC操作的成本
- 业务活动结束时confirm或cancel操作的执行成本
- 业务活动日志成本
- **适用范围**
- 强隔离性、严格一致性要求的业务活动
- 适用于执行时间较短的业务（比如处理账户、收费等业务）

柔性事务解决方案：TCC（两阶段型、补偿型）

用到的服务模式

- TCC操作、幂等操作、可补偿操作、可查询操作

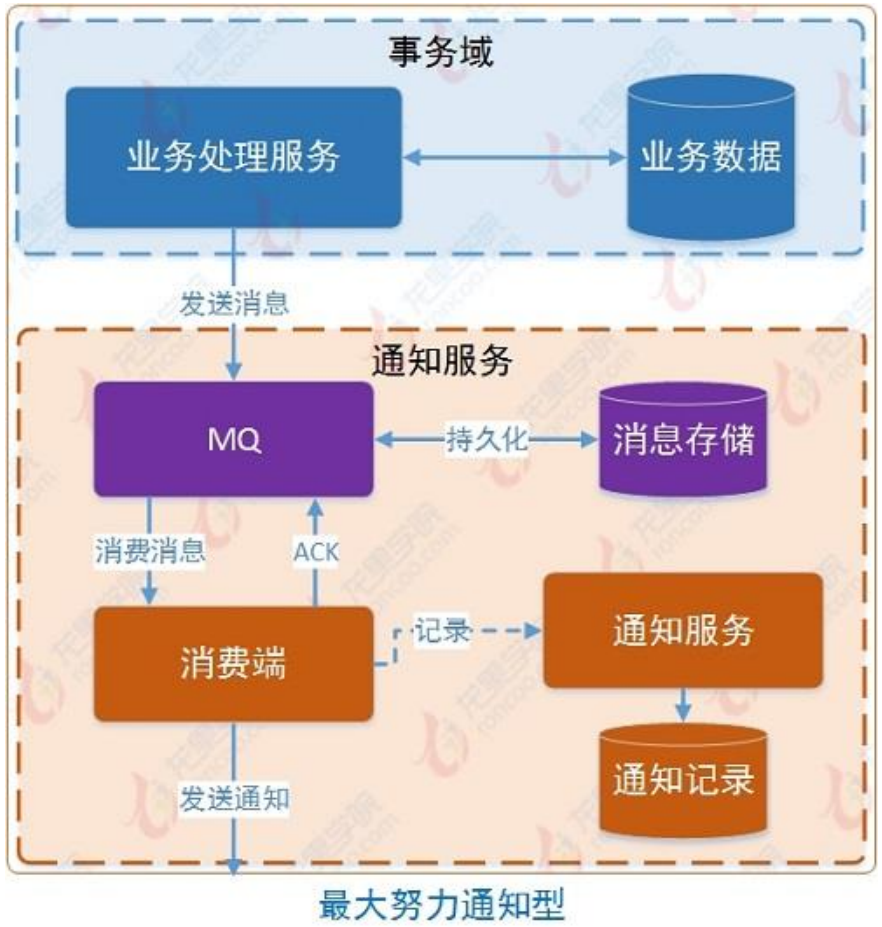
方案特点

- 不与具体的服务框架耦合（在RPC架构中通用）
- 位于业务服务层，而非资源层
- 可以灵活选择业务资源的锁定粒度
- TCC里对每个服务资源操作的是本地事务，数据被lock的时间短，可扩展性好（可以说是为独立部署的SOA服务而设计的）

行业应用案例

- 支付宝XTS（蚂蚁金融云的分布式事务服务DTS）

柔性事务解决方案：最大努力通知（定期校对）



最大努力通知型

- **实现**
- 业务活动的主动方，在完成业务处理之后，向业务活动的被动方发送消息，允许消息丢失。
- 业务活动的被动方根据定时策略，向业务活动主动方查询，恢复丢失的业务消息。
- **约束**
- 被动方的处理结果不影响主动方的处理结果
- **成本**
- 业务查询与校对系统的建设成本
- **适用范围**
- 对业务最终一致性的时间敏感度低
- 跨企业的业务活动

柔性事务解决方案：最大努力通知（定期校对）

用到的服务模式

- 可查询操作

方案特点

- 业务活动的主动方在完成业务处理后，向业务活动被动方发送通知消息（允许消息丢失）
- 主动方可以设置时间阶梯型通知规则，在通知失败后按规则重复通知，直到通知N次后不再通知
- 主动方提供校对查询接口给被动方按需校对查询，用于恢复丢失的业务消息

行业应用案例

- 银行通知、商户通知等（各大交易业务平台间的商户通知：多次通知、查询校对、对账文件）

总结

- **常用的分布式事务解决方案**

- 刚性事务
  - 全局事务（标准的分布式事务）
- 柔性事务
  - 可靠消息最终一致（异步确何型）
  - TCC （两阶段型、补偿型）
  - 最大努力通知（非可靠消息 、定期校对）
  - 纯补偿型（略）

本节课的内容，将作为后续课程实现方案的指导，实现方案的过程中回头看理论，就会很好理解。

（下一节课开始进入解决方案的具体实现课程）

谢谢

THANK YOU

技术支持： Along、Hugo、Peter

龙果学院官方微信公众号