**分布式事务一致性的那些事儿**

数据库本地事务

何为事务？大多数开发人员对事务并不陌生，它是由中间件提供一种机制，通过这种特有的机制将一个活动所涉及的全部操作当做一个不可分割的执行单元，只有这个执行单元的所有操作均能正常执行的情况下才能提交，否则，只要其中任一一个操作执行失败，都将导致整个执行单元回滚。现在的关系型数据库、部分消息中间件都具备这样的事务处理能力。

ACID特性

Atomicity ---原子性，是说事务中的所有操作，要么全部成功，要么全部失败，不会存在中间状态结果。事务在执行过程中如果发生了错误，那么事务会恢复到开始前的状态，就像这个事务从来没有发生过一样。

Consistency ---一致性，是指在一个事务执行前后数据都必须处于一致的状态。如果事务成功执行，那么系统中所有变化将正确地应用，系统处于有效状态。如果在事务中执行出错，那么系统中的所有变化将自动地回滚，系统恢复到原始状态。

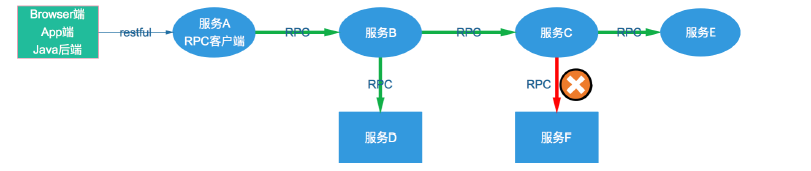
Isolation---隔离性，是指在多个请求并发操作相同的数据时，每个事务都有各自的完整数据空间。由并发事务所做的修改必须与任何其他并发事务所做的修改隔离。事务查看数据更新时，数据所处的状态要么是另一事务修改它之前的状态，要么是另一事务修改它之后的状态，事务不会查看到中间状态的数据。

Durability---持久性，是指当事务成功结束，它对数据库所做的更新就必须永久保留。即使发生系统崩溃，重新启动数据库系统后，数据库还能恢复到事务成功结束时的状态。

ACID特性是事务所具备基本特性， 目前关系型数据库大多实现了事务功能。传统单体应用中，对数据库的访问控制在一个数据库内，一个数据库事务操作不存在跨数据库访问，通过数据库本身提供的事务能力，很好的解决了一致性问题

分布式事务及一致性问题的产生

本地事务可以很好的解决一致性问题，然而事情却有了些变化。随着分布式及微服务架构越来越盛行的今天，越来越多的企业都在进行微服务改造，希望将自身的大而杂的几个大应用分离细化成数量众多的微小服务，尽量让每个微服务只做自己相关的业务，只操作跟自己相关的资源。如图所示，一个完整的业务，



分布式服务

被拆分成了ABCDEF六个服务，每个服务有独立的进程空间、独立的数据库、单独开发部署，服务之间基于RPC调用，每个服务依赖数据库的本地事务能力，控制业务本身的数据一致性，但是在这个调用过程中，如服务C调用F异常，导致服务ABC的业务数据回滚(因调用异常导致本地事务回滚)，F的业务数据可能已提交或者未执行，服务E本地事务已提交，而服务D还未调用，这几个服务的数据状态存在不一致的情况，部分成功部分失败。但从全局来看，使用者肯定不希望这样的发生，使用者希望的是如果成功那大家的数据都提交，但是如果某个服务失败，那已经提交的数据也必须回滚。

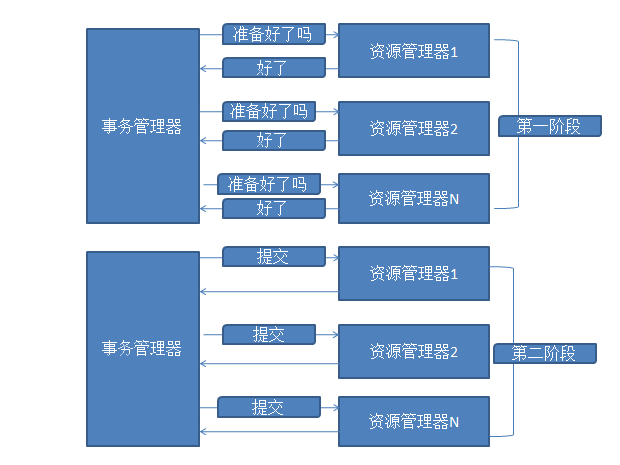
分布式事务一致性解决方案

目前分布式事务解决方案也有多种，基于两阶段提交的XA分布式事务和满足Base特性的最终一致性方案。

两阶段提交协议与XA事务

两阶段提交协议（2PC）可以保证数据的强一致性。它是协调所有分布式原子事务参与者，并决定提交或取消的分布式算法，同时也是解决一致性问题的一致性算法。

XA事务是基于两阶段提交协议的一个分布式事务实现，XA事务主要定义了事务管理器和资源管理器之间的接口，XA接口是双向的系统接口，在事务管理器以及一个或多个资源管理器之间形成通信桥梁。



基于两阶段提交的XA事务

在XA事务中，由事务管理器协调多个资源管理器，在第一阶段：事务管理器给每个参与者(资源管理器)发送Prepare消息，每个参与者要么直接返回失败，要么在本地执行事务，写本地的redo和undo日志，但不提交，到达一种“万事俱备，只欠东风”的状态。第二阶段：如果事务管理器收到了参与者的失败消息或者超时，直接给每个参与者发送回滚(Rollback)消息；否则，发送提交(Commit)消息；参与者根据事务管理器的指令执行提交或者回滚操作，释放所有事务处理过程中使用的锁资源。

XA事务优点

1) 实现了事务的隔离，确保了强一致性，即本地事务未提交的数据对其它事务不可见，事务要么都提交成功要么都失败

2) 业务编程简单，由于事务管理是由事务管理器及本地事务资源管理器实现，开发者必须要介入太多事务相关的工作

XA事务缺点

1) 同步阻塞调用，在事务执行过程中，所有参与者同步锁定资源以实现隔离，被锁定的资源长时间不能被其他事务访问

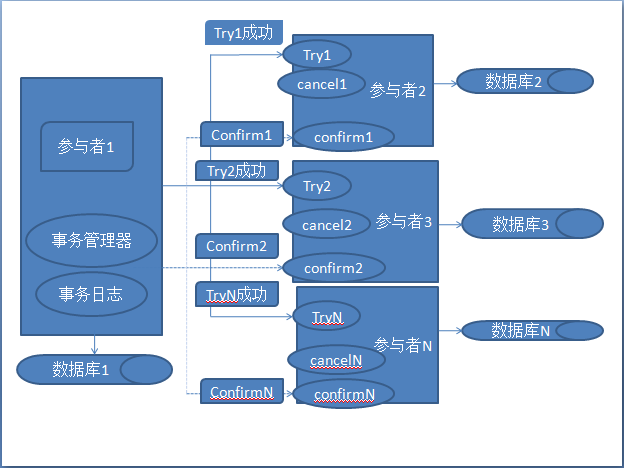
2) 需要本地事务支持，即本地数据库需要支持XA协议

3) 需要有事务管理器统一协调资源管理器，事务管理器本身存在单点问题

4) 事务会出现无法确认的状态。当事务管理器发出commit请求后，然后事务管理器宕机，而参与者可能只有一个已提交，其它参与者尚未提交，此时如果唯一的参与者也宕机，整个事务状态将无法确认

TCC事务

TCC事务即Try-Confirm-cancel三个部分，是柔性事务的一种，实现的是最终一致性，适合于同步调用过程。TCC是应用层的两阶段提交，不需要事务本地数据库对XA协议的支持，TCC事务模型由事务管理器TM（TransactionManager）和资源管理器RM（ResourceManger）即参与者组成。



TCC事务

事务管理器TM：

负责TCC全局事务发起、边界管理、事务提交或回滚等，包括记录维护TCC全局事务的事务状态和每个参与者的子事务状态，事务管理器可以单独是一个服务或者以模块的方式内置于参与者中。整个分布式事务由多个参与者组成，参与者之间通过RPC调用参与者的业务方法Try，第一个参与者负责担当起TM的角色，当参与者之间Try方法都成功后，TM对其余参与者执行Confirm调用，如果Try方法调用不成功，则对其余参与者执行Cancel调用。

参与者RM：

参与者负责提供TCC业务操作，是整个事务的操作方。参与者必须实现Try、Confirm和Cancel三个方法，供其他参与者调用Try业务方法，并由事务管理器调用器Confirm或Cancel方法。由于Confirm和Cancel操作可能被重复调用，故要求Confirm和Cancel两个接口必须是幂等的。

Try：尝试执行业务

1) 完成所有业务检查(一致性)

2) 预留必须业务资源(准隔离性)

3) 基于中间状态执行资源处理

Confirm：确认执行业务，只使用Try阶段预留的业务资源

1) 真正执行业务

2) 将中间状态的资源置为最终状态

Cancel：取消执行业务

1) 释放Try阶段预留的业务资源

2) 将中间状态的资源恢复

TCC事务异常处理与人工处理

Try异常：事务管理器发起Cancel，各参与者执行撤销操作

Confirm&Cancel异常：导致该参与者事务无法确认，其数据在短时间内处于不一致状态。因此，需要人工介入，通过全局事务人工控制台，对不一致的全局事务可进行重试操作，让其最终保证一致性。

TCC事务优点

1) 解决了跨应用的事务问题

把数据库层面的两阶段提交提到应用层来实现，不需要数据库层面来支持XA协议，规避了数据库的XA支持的缺陷

2) 由于参与者业务完成后立即执行了本地事务提交，不存在长时间锁定资源问题，性能也相对提高

TCC模式缺点

1) 需要从业务的角度来设计业务接口，确保业务可分解成Try、Confirm、Cancel三个阶段，增加了业务编程的复杂性

2) 由于每个应用的网络不一定可靠，可能会多次调用业务接口，需要业务层面考虑幂等性操作

3) 由于三个阶段都是同步调用过程，因此随着参与者增多，对主业务响应速度有影响

用友TCC分布式事务方案

用友微服务治理平台分布式事务TCC方案基于用友RPC框架，通过在项目中引入tcc框架后自动创建本地事务管理器依赖的事务表，在服务接口上简单使用一个TccTransactional注解即可注入TCC事务功能

用友TCC分布式事务方案特点

1) 完全基于用友RPC框架插件机制，对代码无侵入

2) 无事务中心，将事务管理器的功能以模块方式织入到参与者，减少事务参与者与事务管理器的网络通信，由某个参与者充当事务管理器角色

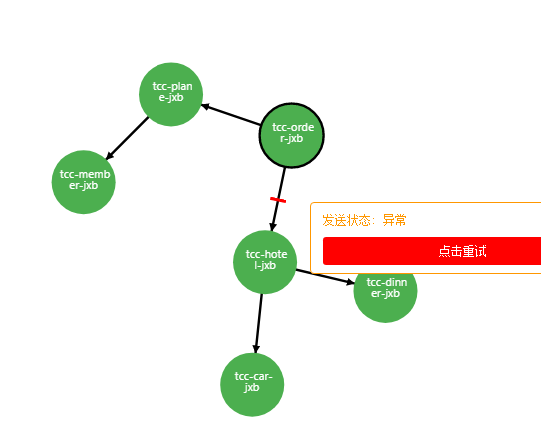
3) 事务信息与业务同库

4) 针对确认或取消异常的统一人工控制台与重试机制

5) 支持mysql、oracle及sqlserver等数据库



全局事务人工控制台



事务重试

基于可靠消息的最终一致性

可靠消息事务最终一致性方案核心思想是用MQ作为服务之间的消息代理，使服务之间解耦，当服务A需要与服务B协作时，服务A执行完其自身业务逻辑后向服务B发送消息通知，服务A发送消息到队列之前，先将消息记录到本地发送表，此操作与服务A的业务处于同一个本地事务中，本地事务提交后，服务A返回本次操作，然后由定时器将待发送的消息记录发送到MQ，服务B接收到消息后执行其业务逻辑，在同一本地事务中操作业务数据并插入事务接收记录，最终完成整个流程。基于可靠消息的分布式事务最终一致性涉及到以下几个组件， 

1)本地事务管理器

业务本地数据库，确保同一个事务内的多个数据操作的ACID特性。

2)本地消息表

记录消息发送和接收及错误信息，每个消息有一个全局唯一的消息ID，消息发送表和消息接收表以此为唯一索引。

3) 消息中间件：

负责消息转发，通过其ACK特性确保消息一定发送或接收成功。

4) 定时器

定时将待发送的消息记录发送到消息队列，直到发送成功为止。

5) 监听器

实时监听消息队列消息，当接收到消息后处理业务，并将接收到的消息记录到本地消息接收表。

基于本地消息记录与消息队列的可靠消息分布式事务最终一致性方案有效解决了以下几个问题

1) 如何保证消息服务A业务执行成功，消息发送失败的情况？

由于是本地消息记录，充分利用了本地事务的ACID特性，服务A将业务数据操作与消息发送记录控制在同一个本地事务中，服务A的业务数据操作和插入发送消息记录会同时成功或失败，不存在只有一个成功的情况

2) 如何保证消息一定会发送？

利用MQ的消息发送确认机制，确保消息一定会至少发送一次。消息发送记录插入发送表时状态是待发送，定时器将待发送的记录定期发送至MQ，MQ将消息落盘后向发送端ACK，发送端将消息记录标记为已发送，这个过程中可能会重复发送消息。

3) 既然消息可能重复发送，那如何保证消息的幂等性呢？

每条消息有一个全局唯一的MSGID，且消息接收方的消息接收表以此为唯一索引，消息接收方接收到消息时，处理业务数据并插入接收到的消息，利用数据库的唯一性约束规则，确保只有一次能成功插入消息接收记录，确保业务数据操作也只有一次操作成功。

4) 消息无法发送到消息队列怎么办？

消息发送定时器向MQ发送消息异常时将异常信息登记，超过一定次数则将消息标记为失败，并上报到人工控制台，通过人工控制台可以对消息忽略或重试，消息接收失败时也如此处理，确保最后一定能执行成功。

用友可靠消息事务一致性框架

基于本地消息记录与消息队列的可靠消息事务最终一致性方案需要解决的是将消息记录与业务操作在同一个本地事务中，这就需要参与业务的服务自己建里消息记录表，上游业务完成的同时需要将业务消息记录插入到本地消息记录表，且需要控制两部分操作在同一个本地事务中，还需要自行去扫描发送消息记录表，将待发送的消息记录发送到MQ。而作为消息接受服务，需要监听消息自己的MQ队列，接收到消息后处理业务需要确保消息幂等性，如果每个服务都要做一遍这些公共的工作自然是得不偿失，且低效冗余的，而用友可靠消息分布式事务框架则是专门为此而诞生的一个灵活而高效的一致性框架，使用简单方便，通过简单配置和基于rpc的注解和原生接口即可轻松实现基于本地消息记录与消息队列的分布式一致性，解决了分布式中数据不一致这个难题。可靠消息事务一致性框架具有如下特点：

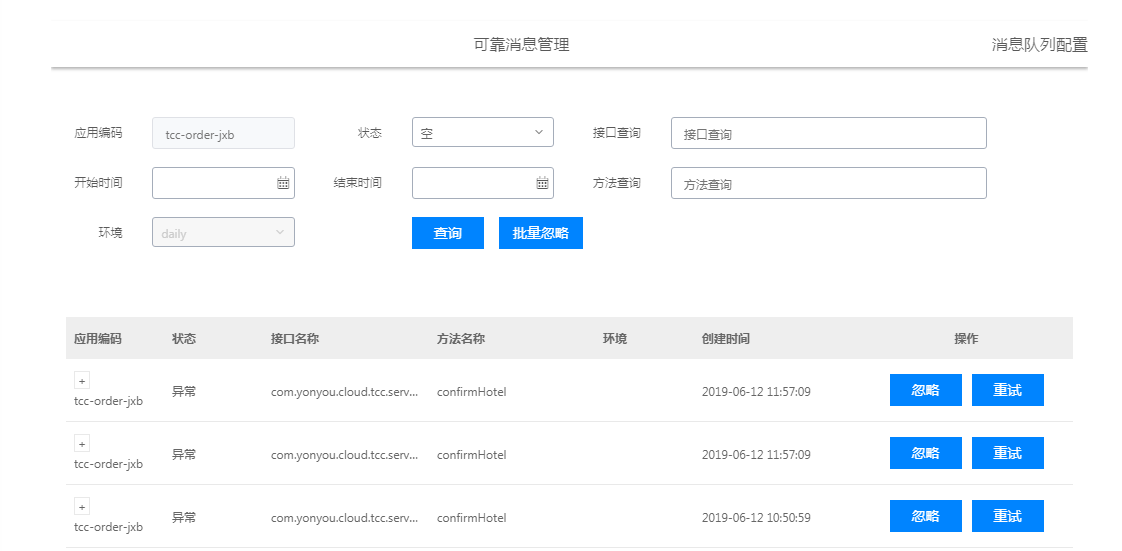
1) 引入框架配置好数据源后，自动在业务库中建立业务日志表

2) 支持多种数据库，Mysql、Oracle、SQLServer及Postgresql

3) 一个注解即可完成服务之间异步消息传递，且保证服务收发消息与业务处理处于同一个本地事务中

4) 保证了消息传递的幂等性，确保一个消息有且只有一次能成功接收并处理

5) 提供人工控制台，服务管理者可以对消息收发异常的消息进行重试操作，重试不成功继续重试，直到成功为止



可靠消息控制台

6) 可针对每个服务提供者设置MQ，目前支持RabbitMQ

7) 实现了RPC框架可靠的异步调用

可靠消息分布式事务实践

通过可靠消息框架实现分布式事务最终一致性的关键是将业务之间同步调用解耦，服务之间不再实时依赖互相调用的结果，而是设计成服务之间异步消息通知模型，然后通过RPC框架提供的SDK实现服务之间的异步调用以达到异步消息通知并执行业务逻辑。