数据一致性框架 YTS

架构设计文档

文档编号：2020-04-28-V1
创建日期：2020-04-28
最后修改日期：2020-09-25
版本号：v1.01
电子版文件名：数据一致性框架-架构设计文档

文档修订摘要

日期	修订号	描述	著者	审阅者
2019-11-25	1	初版+场景梳理	逯永收	
2020-04-28	2	重新整理提纲	刘建民	
2020-05-06	3	增加异常场景章节提纲	刘建民	
2020-05-21	4	更新功能架构和技术架构图	刘建民	
2020-09-23	5		干洪	
2020-09-27	6	调整文档格式以及目录	逯永收	

目录

1 概述

1.1 目的

此文档主要介绍用友云 iuap 平台下，基于微服务架构设计下的分布式事务框架的详细设计，包含对功能的架构设计，各组件间的关系，监控服务，部署结构，兼容性设计，数据结构设计等方面。

编写本文档旨在方便使用 iuap 为基础的微服务架构下，分布式事务框架的相关架构师和工程师、运维人员、测试人员等快速了解平台内部的核心概念，理解各组件间的逻辑关系，并了解内部的核心工作流程及实现方式。

1.2 阅读对象

此文档将适合以下人员阅读：

- 业务领域使用 iuap 数据一致性框架及微服务 SDK 的架构师和工程师；
- 微服务数据一致性相关的架构师和工程师；
- 技术中台的运维和部署人员；
- YS 公有云、用友各领域云相关开发和设计人员；
- 其他和微服务治理平台生态相关的人员；

1.3 设计原则与前提

本框架目前针对 YonSuite（后续支持采购、财务、人力、协同等各领域云）如何保障各个微服务之间的数据一致性考虑，目前适用场景有限。

框架和 iuap 平台下的 MDD、MDF 及 RPC 结合，结合本地数据库事务提交机制，目前支持 Mysql。后续逐步支持单纯 RPC 方式或者 HTTP client 方式。

技术中台的云端，提供事务监控中心，监控和查看未及时完成的及未正确结束的长事务，支持云端查看错误堆栈、链路拓扑图、下发重试命令等功能。

1.4 名词解释

MDD：

模型驱动开发，一种开发模式，YTS 框架适配 MDD 开发框架方式运行。

YTS-SDK：

iuap 微服务架构下，数据一致性框架提供的 java SDK。

定时任务：

处于 YTS-SDK 或者后续的 HTTP client SDK 中的定时任务或者线程。

规则（rule）：

MDD 概念中的规则，一个规则对应一系列的代码执行操作。

回滚：

在参与分布式事务的远程调用或者本地调用报错后，反向操作或者取消操作。

重试：

本文特指在调用回滚逻辑报错后的云端下发命令后的业务重试操作。

事务回调：

基于 Spring 的数据库事务提交回调机制。

注册中心：

微服务架构中的服务地址登记和注销中心，它记录了服务和服务地址的映射关系。在分布式架构中，服务会注册到这里，当服务需要调用其它服务时，就到这里找到服务的地址，进行调用。

RPC 框架：

Remote Procedure Call，即远程过程调用，可以实现客户端像调用本地服务(方法)一样调用服务器的服务(方法)；

序列化：

序列化是一个用于将对象状态转换为字节流的过程，可以将其保存到磁盘文件中或通过网络发送到任何其他程序；从字节流创建对象的相反的过程称为反序列化。

插件：

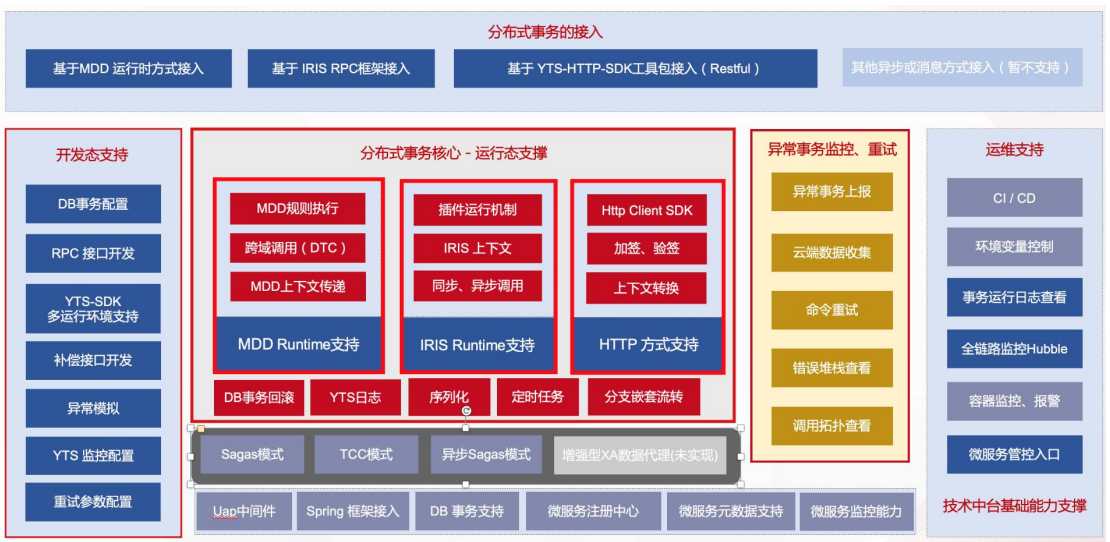
本文档中涉及到的插件是指微服务 SDK 中的执行插件，SDK 中包含启动插件，调用前插件、调用后插件、执行前插件、执行后插件、权限插件、限流插件、链路插件等。

IRIS：

用友云微服务治理平台的 RPC 框架代号。

2 架构设计

2.1 功能架构设计



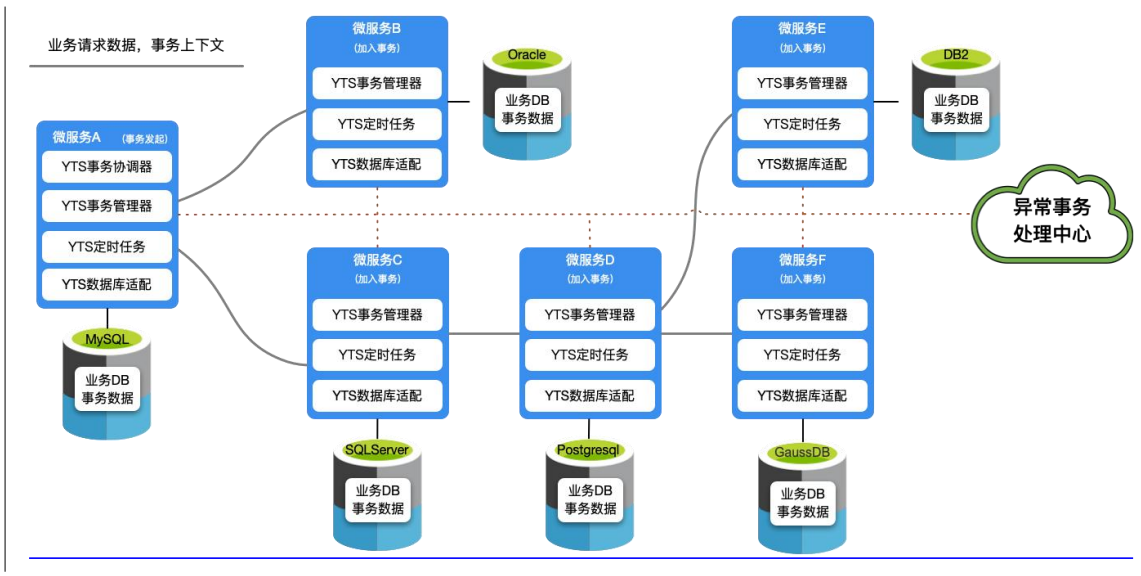
分布式事务框架 YTS 依托于微服务治理平台的 RPC 框架，和 iuap 平台的技术平台紧密结合，统一对外提供数据最终一致性解决方案。

YTS 框架支持多种运行态，包括 MDD、纯 IRIS、HTTP 等；支持多种一致性控制模式，如同步 Sagas、异步 Sagas、TCC 等。不同的运行时和控制模式的组合满足了绝大多数的业务场景需求。

框架依赖很多技术底层技术：数据库本地事务回调、分段的数据库事务日志、定时任务、Spring 框架的事务管理器等等。框架支持多种数据库类型，支持 MySql、SQLServer、Oracle 等主流数据库及达梦等国产数据库。

同时，YTS 框架包含可视化监控及问题排查能力，通过对错误数据的上报，错误堆栈的抓取，全局链路数据的跟踪，可以方便的进行错误事务的拓扑图展示，问题诊断。

2.2 技术架构设计



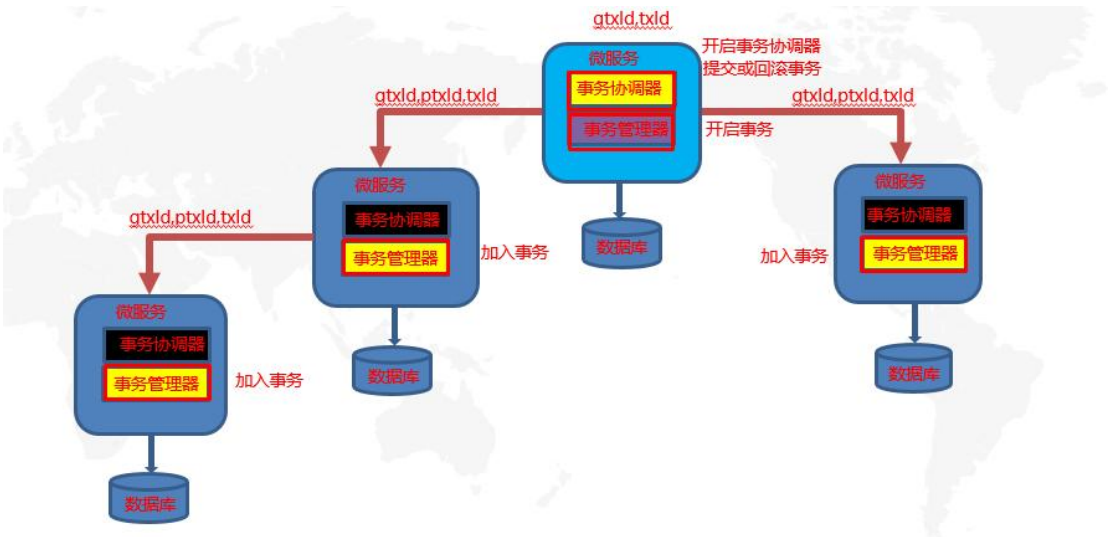
2.2.1 YTS 事务协调器

YTS 事务协调器是基于分布式模式，以模块的形式供应用使用，事务协调器的作用在于对根据全局事务的状态及事务模式，决定是否发起回滚或者是否发起提交。全局事务起点开启事务时，YTS 框架会启动 YTS 事务协调器，并根据事务模式，注册相应的 Spring 事务回调钩子

2.2.2 YTS 事务管理器

YTS 事务管理器负责事务边界管理，负责开启或加入事务，并记录相应的事务日志及调用关系。事务管理器与事务协调器一样，以模块方式提供，当通过 API 开启事务或基于注解或 MDD 规则配置开启事务或加入事务时接入事务管理功能，如果是事务起点，开启事务协调器。

删除[华新]：开启

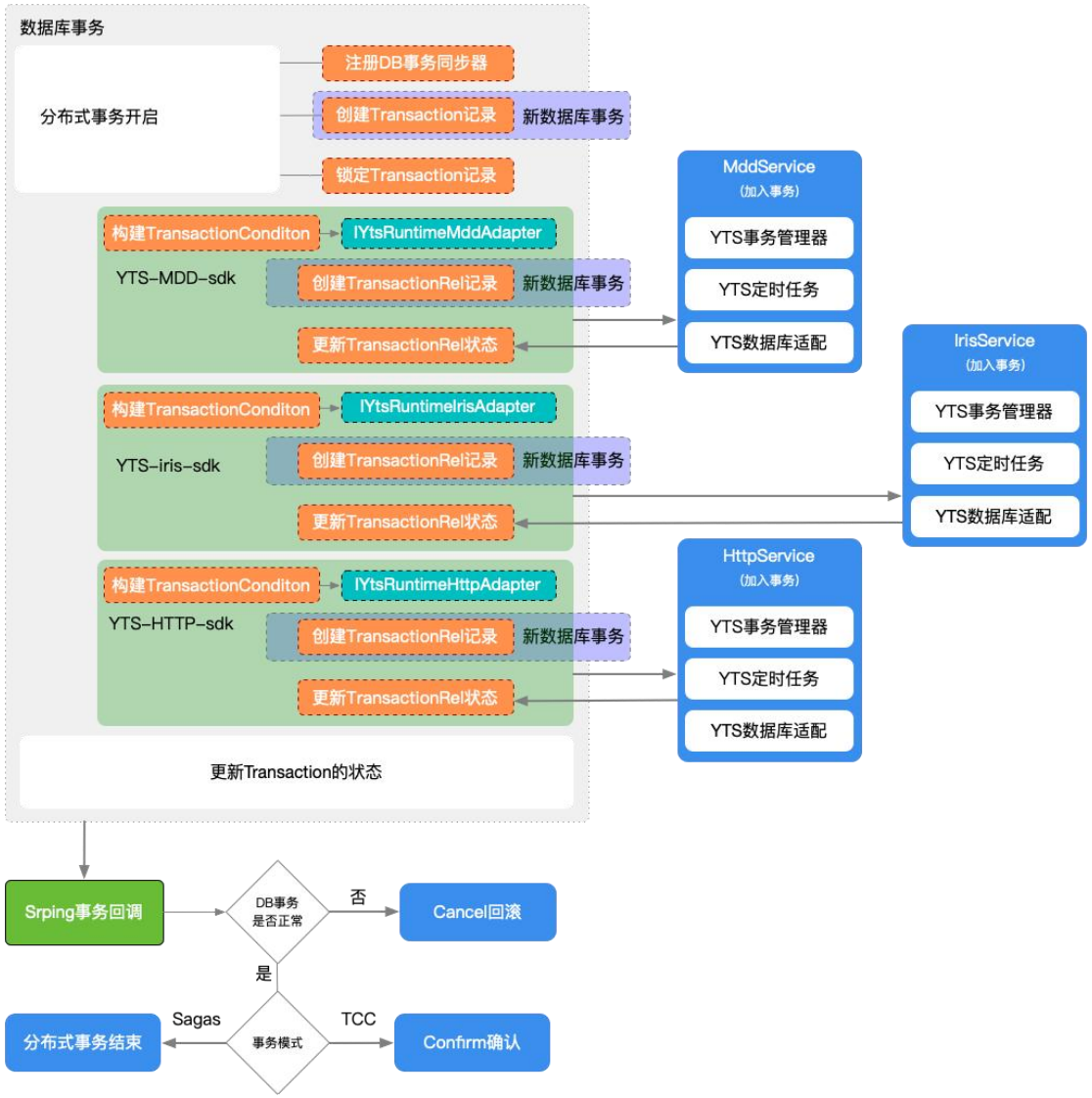


YTS 事务协调器&事务管理器

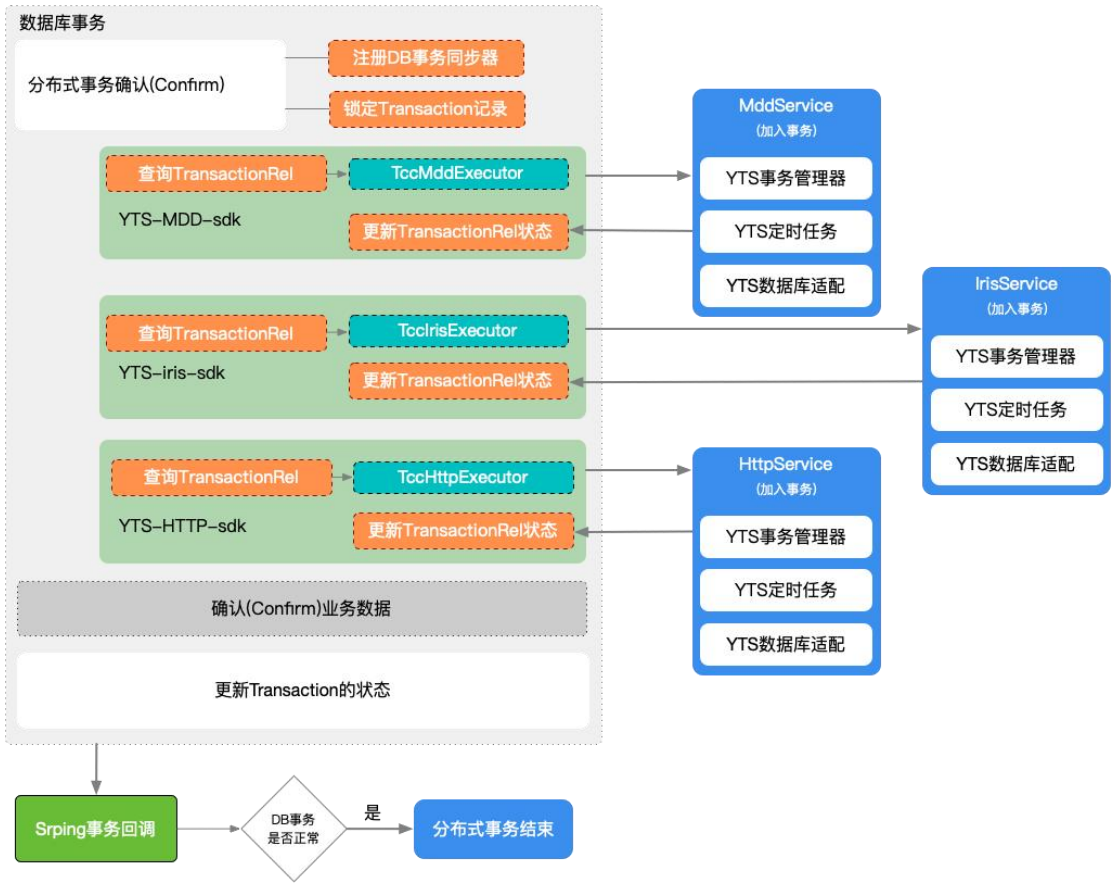
2.2.3 YTS 运行时

删除[华新]:

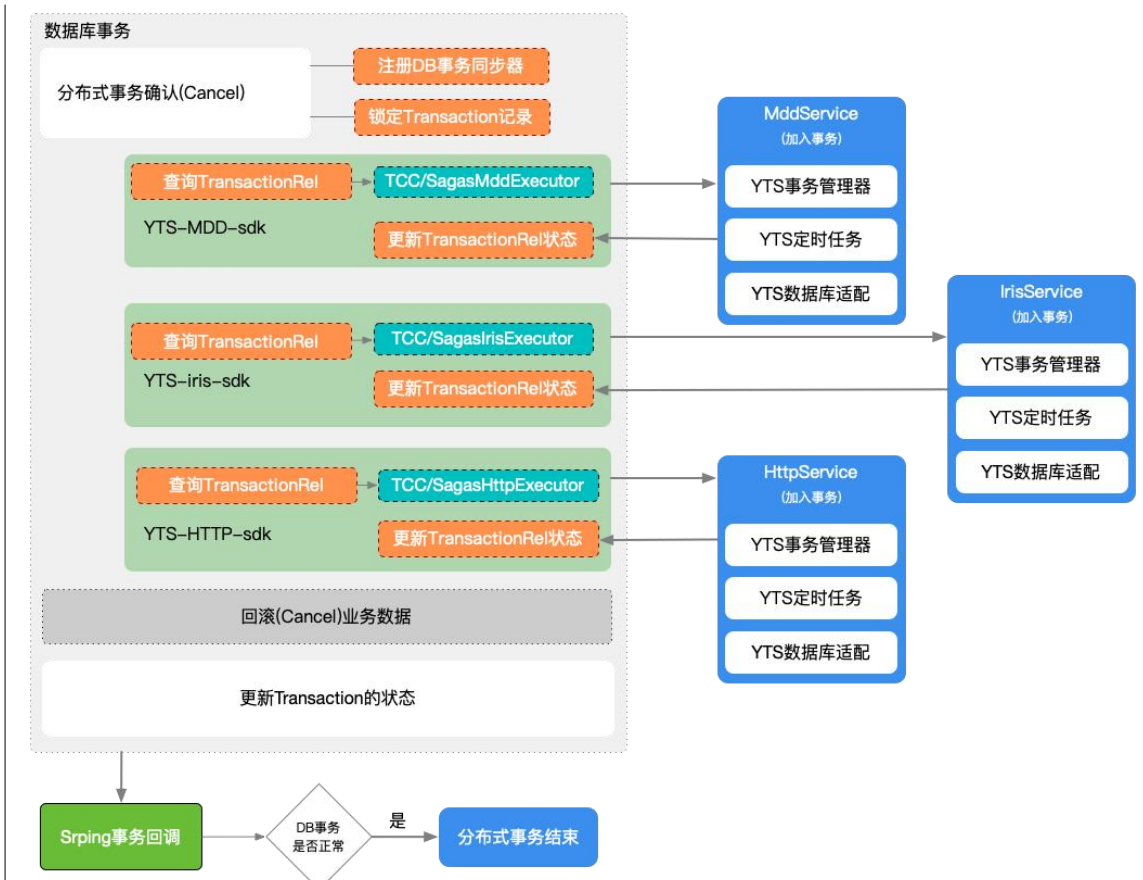
事务第一阶段(Sagas 的 execute，TCC 的 try)



Tcc 模式第二阶段(Confirm)



分布式事务回滚(Cancel)



2.2.4 YTS 框架核心类和枚举

2.2.4.1 Transaction 对象

记录参与长事务的事务节点的信息，主要记录事务节点类型，gtxId(全局事务 ID)，txId(当前事务节点的事务 ID),ptxId(上游事务节点的事务 ID)，以及事务的模式，服务相关信息，接口相关信息，事务状态信息等。

2.2.4.2 TransactionRel 对象

记录事务节点参与本次事务的下游服务的信息，主要记录事务节点信息，gtxId(全局事务 ID)，txId(当前事务节点的事务 ID),ptxId(上游事务节点的事务 ID)，下游服务调用的上下文，服务相关信息，以及下游服务事务状态等信息。

2.2.4.3 事务主状态枚举 YtsStatus

CONFIRMING　　事务初始化状态

CONFIRMED　　事务成功完成

CANCEL_ERROR　　事务回滚失败

CANCEL_SUCCESS　事务回滚成功

ERROR　　参与事务的节点业务操作失败，并且没有调用其他子事务

CANCEL_SUSPEND　没有成功执行本节点事务，但是回滚子事务失败

CANCEL_INIT　　由于超时等业务异常导致回滚操作（cancel）先进行插表的状态

TRY_SUCCESS　　针对 TCC 或异步 Sagas 模式，业务执行成功后的事务 LOG 状态

ACK_FAIL　　异步 Sagas 模式向上 Ack 失败，TransactionRel 状态

ACK_SUCCESS　　异步 Sagas 模式向上 Ack 成功，TransactionRel 状态

2.2.4.4 重试状态枚举 RetryStatus

ERROR　　重试异常

RETRYING　　重试中

RETRY_FAIL　　重试失败

RETRY_SUCCESS　　重试成功

IGNORED　　已忽略

2.2.4.5 事务模式枚举 TransactionMode

SAGAS　　mdd-sagas 模式

TCC　　mdd-tcc 模式

ASYNCSAGAS 异步的 sagas 模式

2.2.4.6 事务节点类型的枚举 TransactionNodeType

ROOT　　根事务节点分布式事务的发起节点

BRANCH　　非根事务节点，加入分布式事务的节点

2.2.4.7 数据上报状态的枚举 ReportStatus

TOUPLOAD　待上传

UPLOADING　　正在上传

UPLOADED　已上传

DOWNED　　已下载

2.2.4.8 冻结状态枚举 FrozenStatus

FROZENUP 已解冻

FROZENIN 冻结中

2.2.4.9 RPC 类型的枚举

MDD mdd 框架

IRIS iris 微服务

HTTP 用友内部基于 Auth-sdk 的 http 方式

IPAAS 支持 ISV 通过 ipaas 平台转发的 http 请求

设置格式[华新]：字体：加粗，字体颜色：自动设置，非加宽量/紧缩量

设置格式[华新]：字体：小四，加粗，字体颜色：黑色，非加宽量/紧缩量

2.2.4.9 YTS 上下文

LogProcessContext

事务管理器的上下文，用来绑定在 spring 容器的事务管理器中，保存当前事务的相关信息如：当前事务调用子事务的关联关系，主事务的状态等

TransactionThreadContext

事务信息的线程上下文，主要用来保存当前线程的 Transaction 的对象，事务的模式等信息

TransactionContext

在参与分布式事务的微服务之间传递分布式事务信息的上下文，主要是在进行 RPC 调用时由上游传输到下游

YtsContext

YTS 组件在 spring 容器内的上下文信息主要保存用户通过 API 保存的用来在正向流程向回滚操作传递的数据

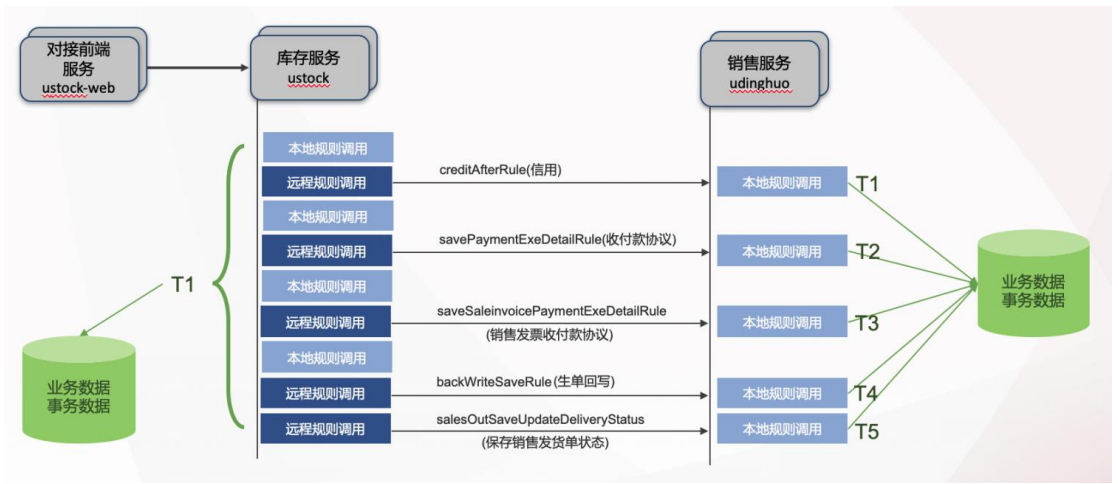
删除[华新]：

设置格式[华新]：行距：单倍行距，正文文本

2.3 逻辑架构设计

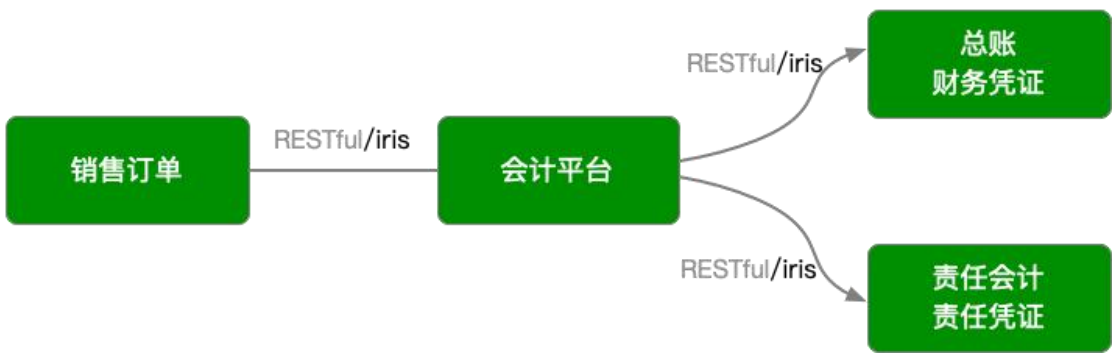
2.3.1 核心业务场景

2.3.1.1 销售出库信息回写（MDD）

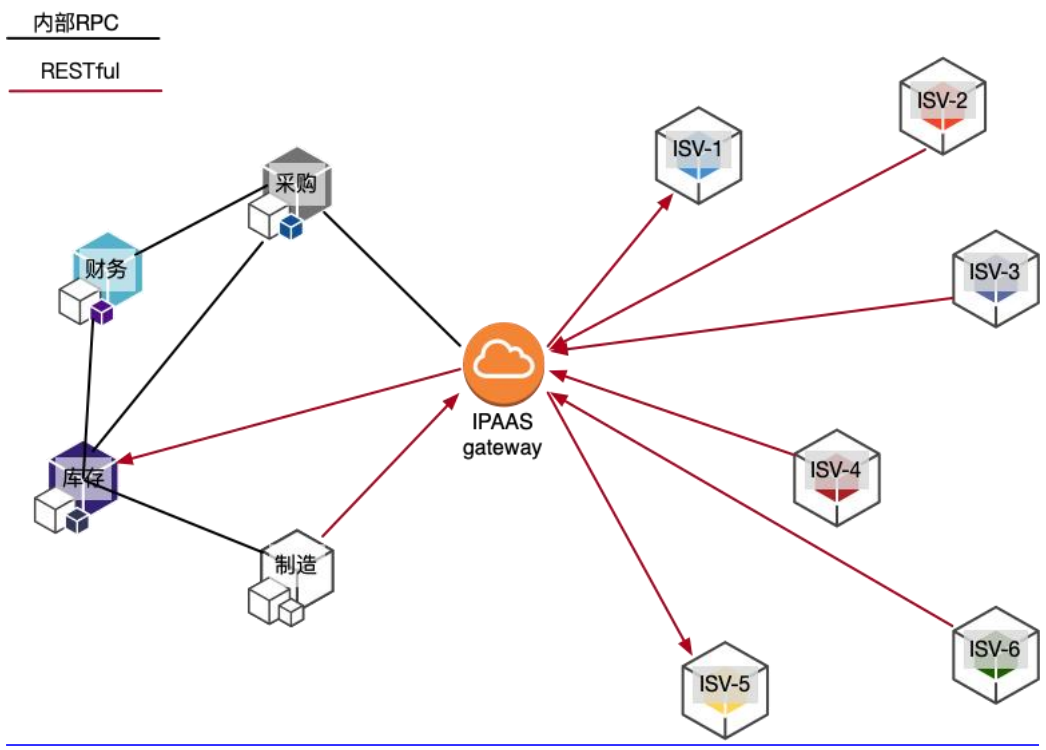


2.3.1.2 财务支付业务凭证（RPC）

财务现有框架为 dubbo 开发并进行了 iris 的对接，现采用 RESTful 进行服务间的交互，可以将 RESTful 接口重构为 iris 微服务的调用。



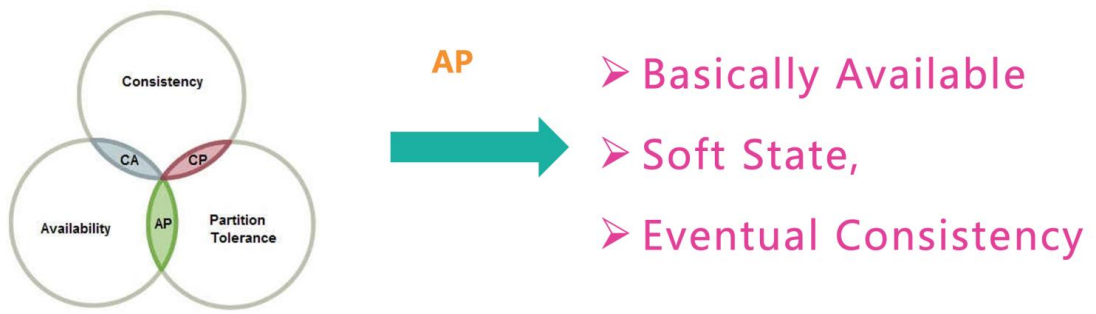
2.3.1.3 生态应用调用用友云开放服务（HTTP）



ISV 厂商以及生态合作伙伴通过开放平台(IPAAS)使用 RESTful 的接口来访问 YonBIP 内部服务，并且内部 YonBIP 通过开放平台(IPAAS)使用 RESTful 接口访问 ISV 厂商以及生态合作伙伴，在需要保证数据一致性的场景使用 YTS 的分布式事务管理来达到数据一致性。

2.3.2 领域概念模型

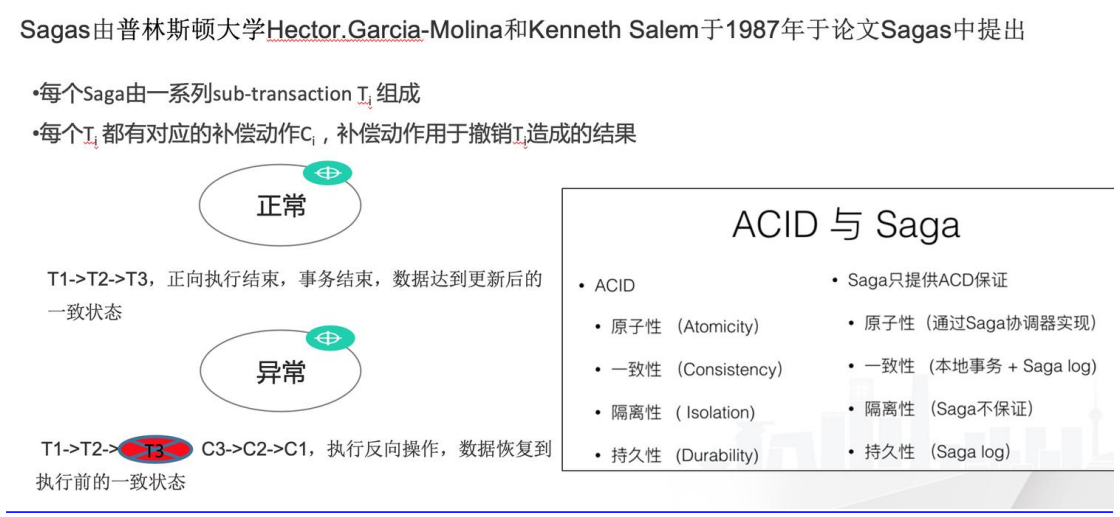
2.3.2.1 YTS 基础理论模型



通过牺牲强一致性来获得可用性，并允许数据在一段时间内是不一致的，但最终达到一致状态

2.3.2.2 基于 BASE 理论的分布式一致性解决方案

2.3.2.2.1 Sagas 模式



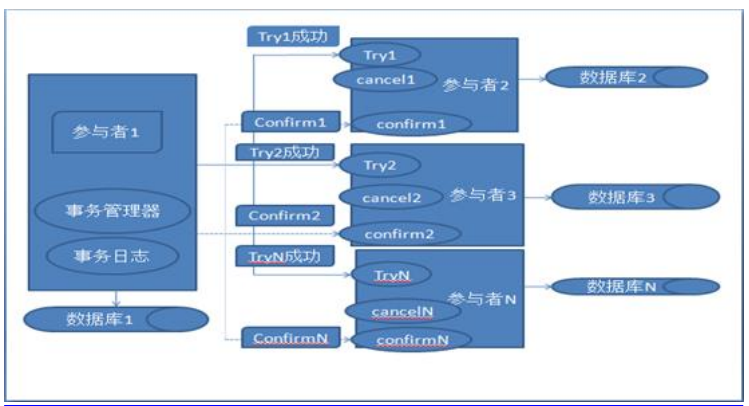
Sagas 模式适用场景：每个参与者均提供反向操作，当正向业务有失败时，由事务协调器调用参与者提供的反向操作

Sagas 模式缺点：不具备隔离性，可能会导致无法回滚的情况

Sagas 模式条件：反向操作一定成功，允许全局事务在一定时间内的不一致

2.3.2.2.2 TCC 模式

TCC 模式把业务的操作从逻辑上划分为两个阶段及 Try 阶段和 Confirm/Cancel 阶段，首先通过 rpc 框架调用各参与者的业务方法（Try），如果都成功则事务管理器调用各参与者的 Confirm 方法，否则只要事务发起者捕获到失败，则调用各参与者的 Cancel 方法



TCC 模型

Try：尝试执行业务阶段

完成所有业务检查（一致性）

预留必须业务资源（准隔离性）

基于中间状态执行资源处理

Confirm：提交阶段

确认执行业务，只使用 Try 阶段预留的业务资源真正执行业务

将中间状态的资源置为最终状态

Cancel：回滚业务阶段

释放 Try 阶段预留的业务资源

将中间状态的资源恢复

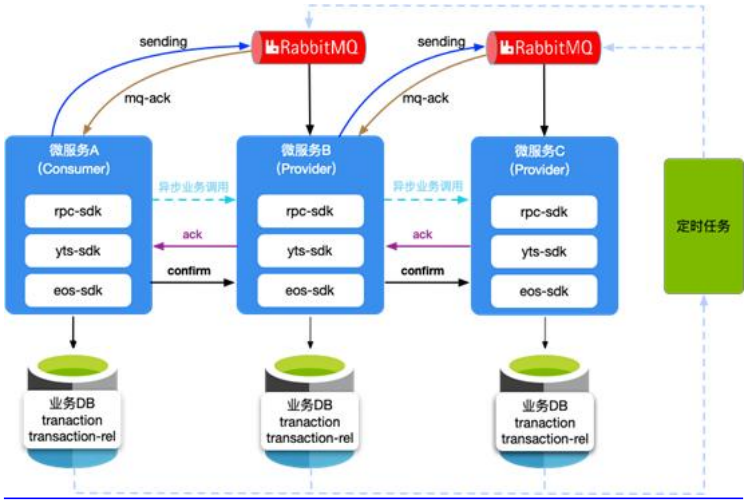
TCC 模式特点：1)由于从业务层拆分成两个阶段，可以通过两阶段模式实现业务资源的隔离，2)每个阶段执行完成后提交本地事务，性能好

TCC 模式缺点：业务侵入性高，对于一些老系统代码改动逻辑大

TCC 模式条件：Confirm/Cancel 操作一定成功，允许全局事务在一定时间内的不一致

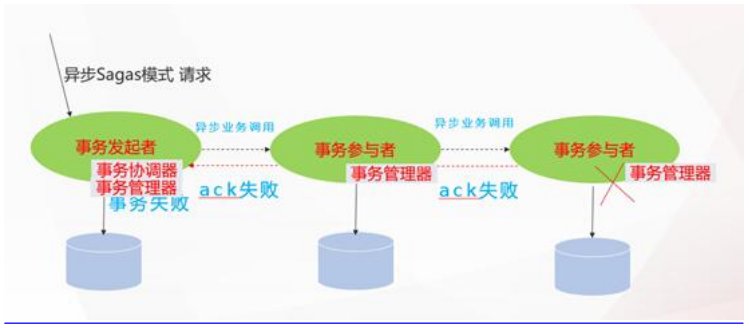
2.3.2.2.3 异步 Sagas 模式

异步 Sagas 模式类似于 TCC 模式，区别在于业务之间的调用通过可靠消息框架来实现了业务的异步调用，异步 rpc 调用不会阻塞调用方的执行流程，调用方不等待异步调用的返回结果，下游业务执行结束后，会向上游 Ack 执行结果，当事务发起者成功收到所有下游的 Ack 结果后，事务协调器根据 Ack 状态决定全局事务的状态，如果都 Ack 成功，则全局事务成功，由事务协调器通知下游全局事务成功，修改事务状态，如果有下游 Ack 失败，则事务协调器只负责修改该全局事务状态为失败，不会立即回滚或重试，而是由人工决定对该事务的处理逻辑，YTS 框架提供了努力向前重试或向后补偿机制。



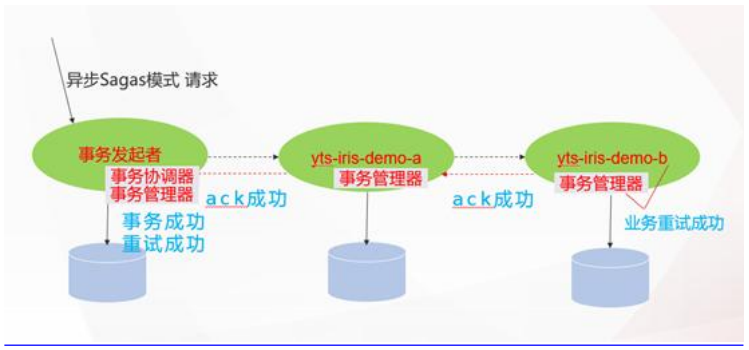
异步 sagas 模式模型

删除[华新]:

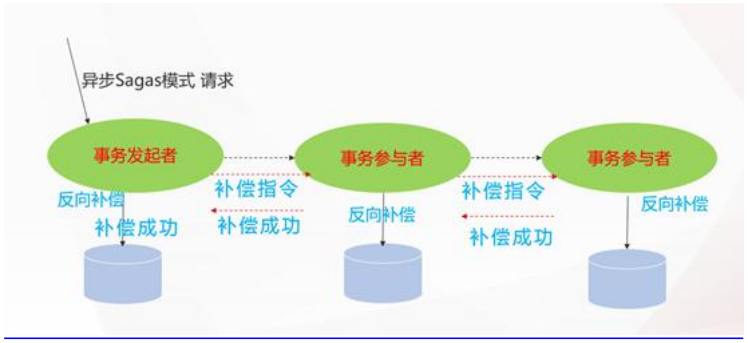


异步 Sagas 模式执行过程

删除[华新]:



重试处理过程



补偿处理过程

异步 Sagas 模式使用场景：通过异步消息进行业务解耦，上游业务需要快速返回前端结果，提高业务入口响应速度

异步 Sagas 模式优点：通过异步解耦，提升响应速度，灵活的异常处理方式，可努力向前也可以向后补偿

异步 Sagas 模式缺点：业务之间异步调用，涉及到业务流程改造及前端交互方式的变化

2.3.2.2.4 基于 MQ 的可靠消息模式

基于 MQ 的可靠消息模式是上游业务完成后，将事件信息通过 MQ 发送下游服务，下游服务接收事件消息，处理下游业务，为了保证消息可靠性，上游业务需要通过本地数据库记录消息发送记录，消息发送记录与业务在一个本地事务，确保业务成功，事件的消息发送记录也保存成功，通过定时任务或其它机制想消息发送到 MQ，下游业务接收消息，处理相应事件,处理成功后向 MQ 确认消息已接收

2.3.2.3 技术支撑设计

2.3.2.3.1 基于 Spring 的 DB 事务回调

正向业务逻辑执行的 DB 事务回调：

正向操作成功后，通过 Spring 的 DB 事务回调来完成修改事务状态，清除事务上下文的操作，TCC 模式下还需要调用下游的确认(confirm)的方法。

正向操作失败时，通过 Spring 的 DB 事务回调来完成异步调用事务的回滚逻辑来回滚下游的子事务来保证下游服务数据的一致性。

回滚阶段的 DB 事务回调：

回滚阶段事务操作成功时修改事务状态为 CANCEL_SUCCESS

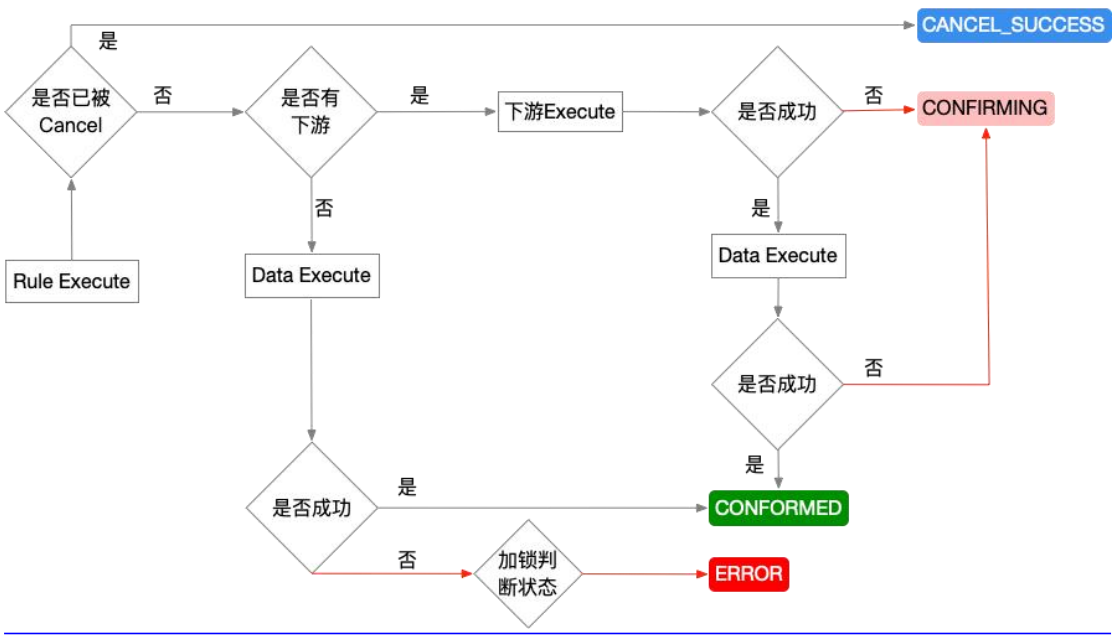
提交阶段的 DB 事务回调：

提交阶段事务操作成功时，修改 TCC 事务模式修改事务状态为 CONFIRMED

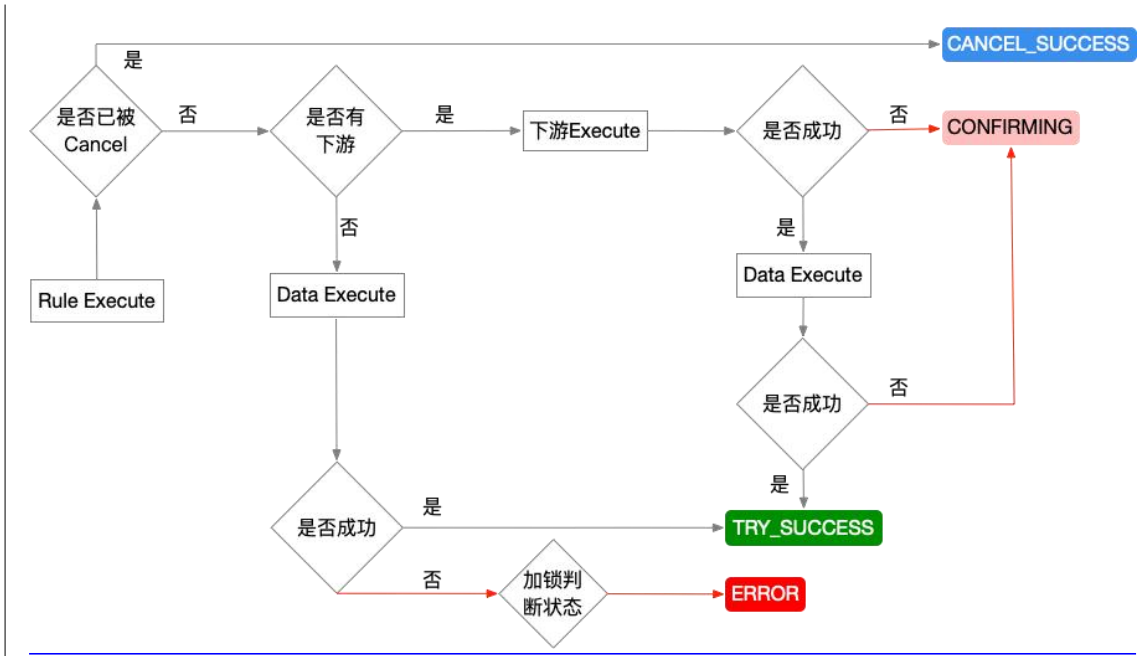
2.3.2.3.2 分布式事务 LOG 及状态转换

删除[华新]：

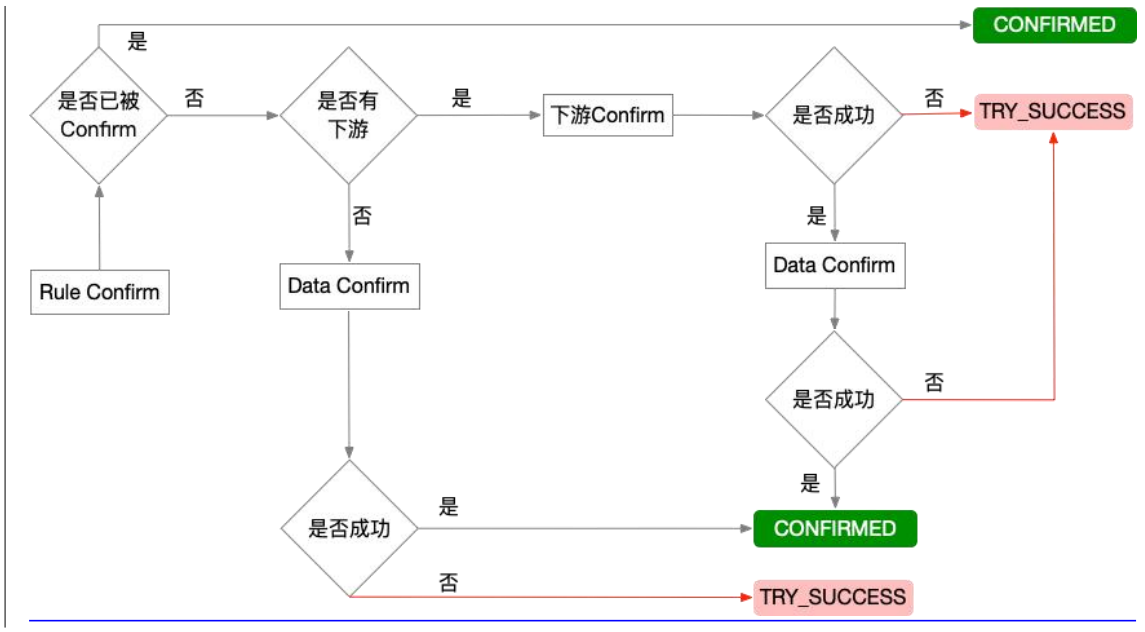
Sagas 正向操作的状态扭转：



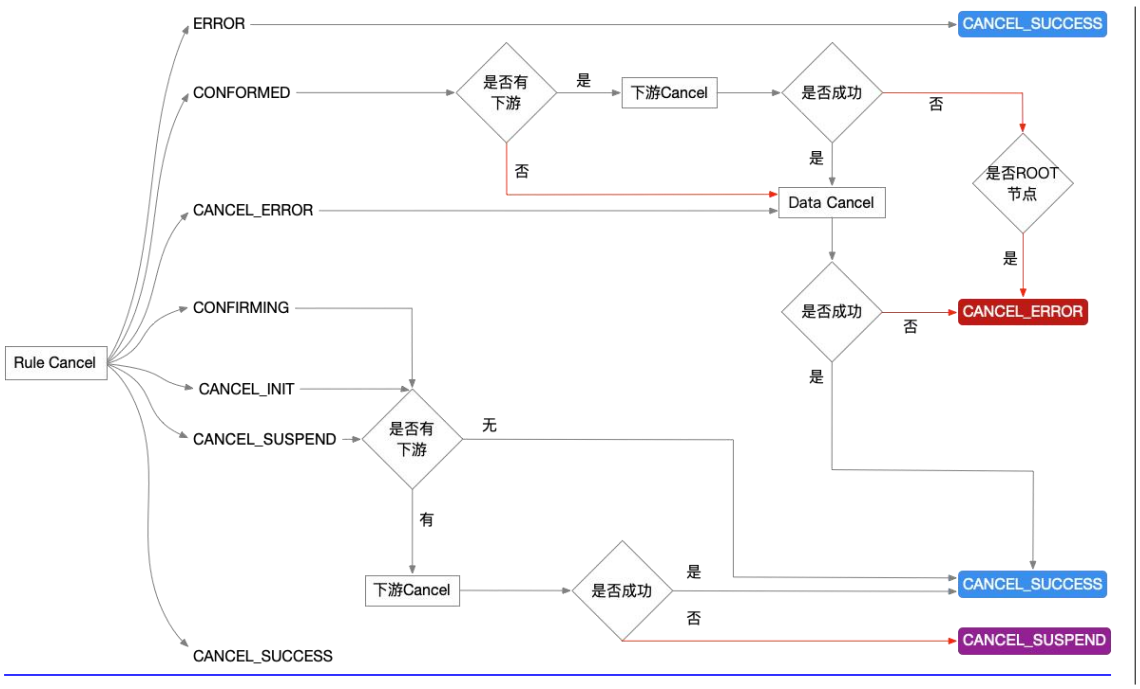
Tcc 模式正向操作(Try)状态扭转：



Tcc 模式确认(Confirm)操作状态扭转：



TCC，Sagas 模式事务异常 cancel 操作的状态扭转：



2.3.2.3.3 序列化机制

RPC 调用上下文序列化采用 Hession 协议的二进制协议，hession 在上下文包含 java 枚举以及泛型操作时能够正确处理。在 RPC 调用前，YTS 的 sdk 将 RPC 调用的上下文信息，持久化到 TransactionRel 的数据表中，在确认(Confirm)，回滚(Cancel)时从 TransactionRel 表中反序列化为 RPC 调用上下文进行 RPC 的调用。

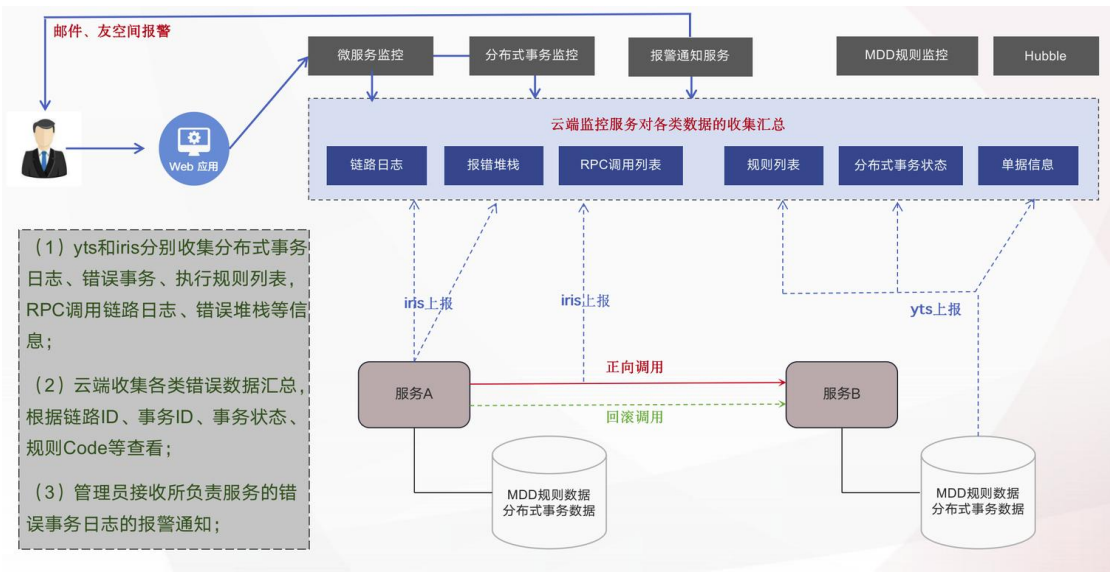
2.3.2.4 统一监控设计

YTS 框架，采用了多种方式的监控结合的方式，进行全局事务的监控和管理。包括错误全局事务日志上报下载，分布式链路数据拓扑展示，错误堆栈上报汇集，MDD 规则信息关联等。

SDK 中，会采集调用链路数据和全局事务日志信息，异步的方式上报到云端，后端支撑服务对数据进行集中存储和可视化展示。

针对错误的事务信息，会通过友空间、邮件等方式进行报警通知。

同时，SDK 会上报全局事务关键环节的处理事件，同 Hubble 链路诊断框架结合，进行诊断报告的关键事务事件的展示。



删除[华新]：

2.3.2.5 定时扫描数据任务

1. 错误上报

   当业务场景中有任何一个节点出现异常，导致整个分布式事务不能正常执行时，将不能正常执行的分布式事务的事务数据上报到云端分布式事务管理中心，进行统一管理。该定时任务还会对重试后的事务状态进行上报，将最新的事务状态同步到云端。

2. 长时间未处理

   当应用没有完成分布式服务的调用而异常退出，或者服务重启会导致分布式事务不能完成，而存在分布式事务挂起的问题。为了解决该问题，会有一个定时任务来扫描长时间没有处理完成的定时任务，然后将事务进行回滚的操作。

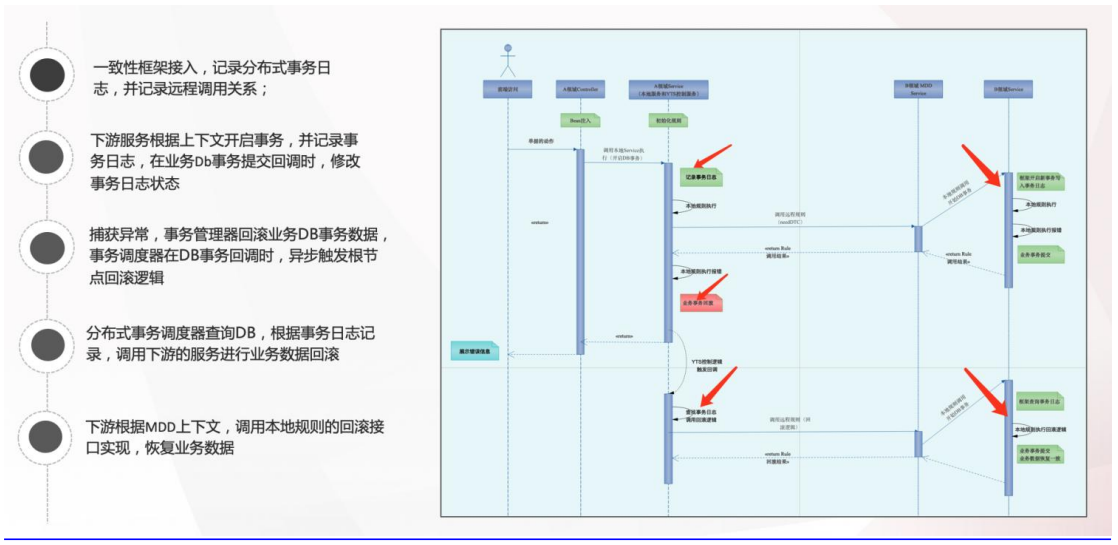
删除[华新]：再

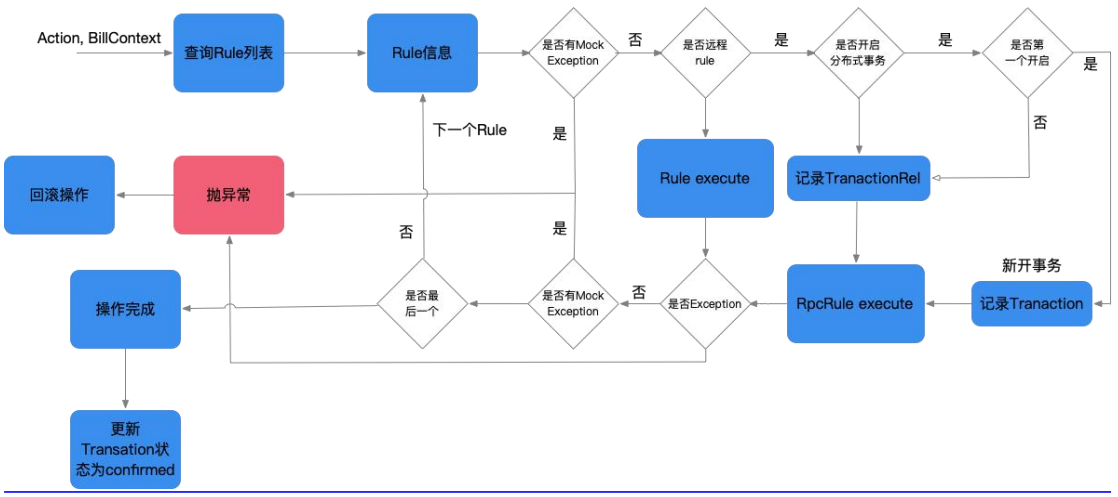
3. 重试
   当用户通过云端的分布式事务管理中心点击了回滚失败事务的重试按钮后，重试任务将点击重试的分布式事务重新进行回滚的操作。
4. 事务数据的拉取
   当用户在云端分布式事务管理中心点击重试后，定时任务将点击了重试的分布式事务的数据拉取到本地，并更新事务的状态为待重试的状态由重试的任务进行重新进行回滚操作
5. 冻结数据解冻

当用户通过冻结 API 将数据冻结后，定时任务会定时查询根节点的事务状态，当根节点事务状态为 Confirmed 时将该事务冻结的数据进行解锁操作。

2.3.3 核心场景及流程

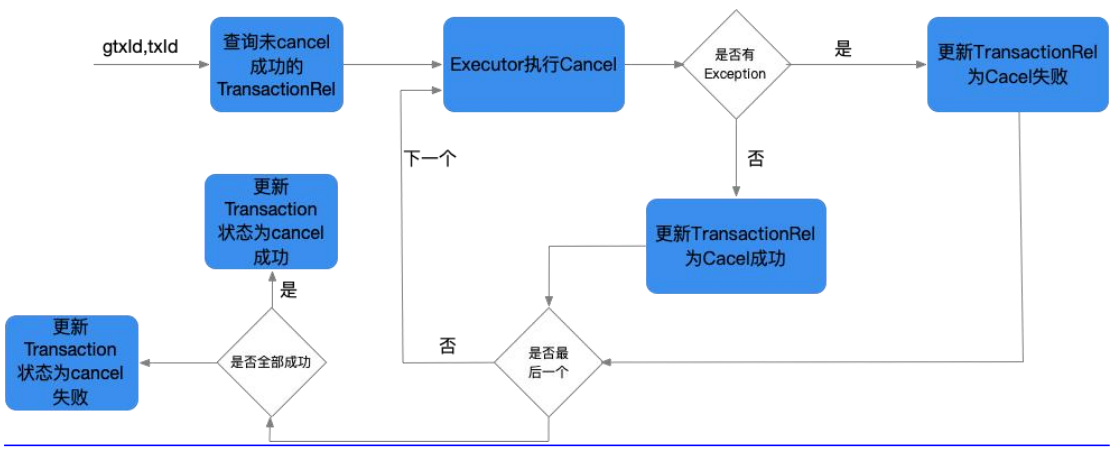
2.3.3.1 MDD 框架正向业务调用





2.3.3.2 正向调用报错后的业务回滚

执行回滚操作时会根据 TransactionRel 的信息，获取对应模式以及框架的 Executor 来执行 Cancel 的方法。



2.3.3.3 回滚报错后的错误信息上报

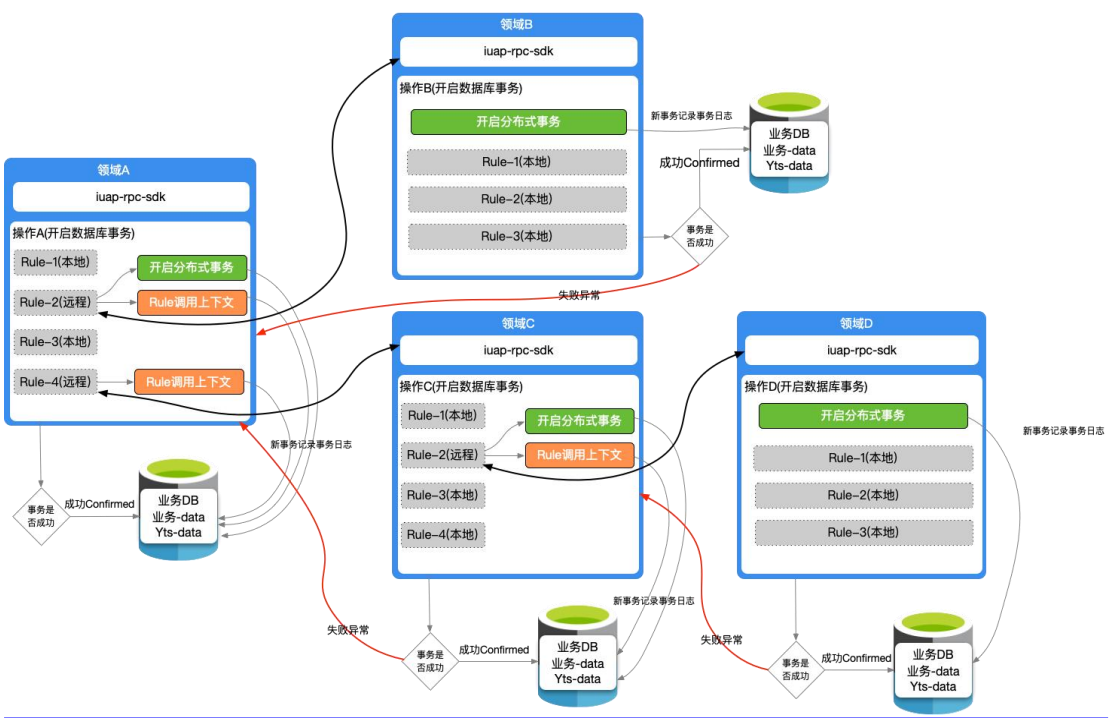
当事务出现异常，并且调用回滚（cancel）操作异常后，事务的状态为 CANCEL_ERROR,定时任务将错误的事务数据上报到云端分布式事务管理中心，并发送报警信息到相应的开发人员的友空间，提醒开发人员。

2.3.3.4 云端重试命令的下发及执行

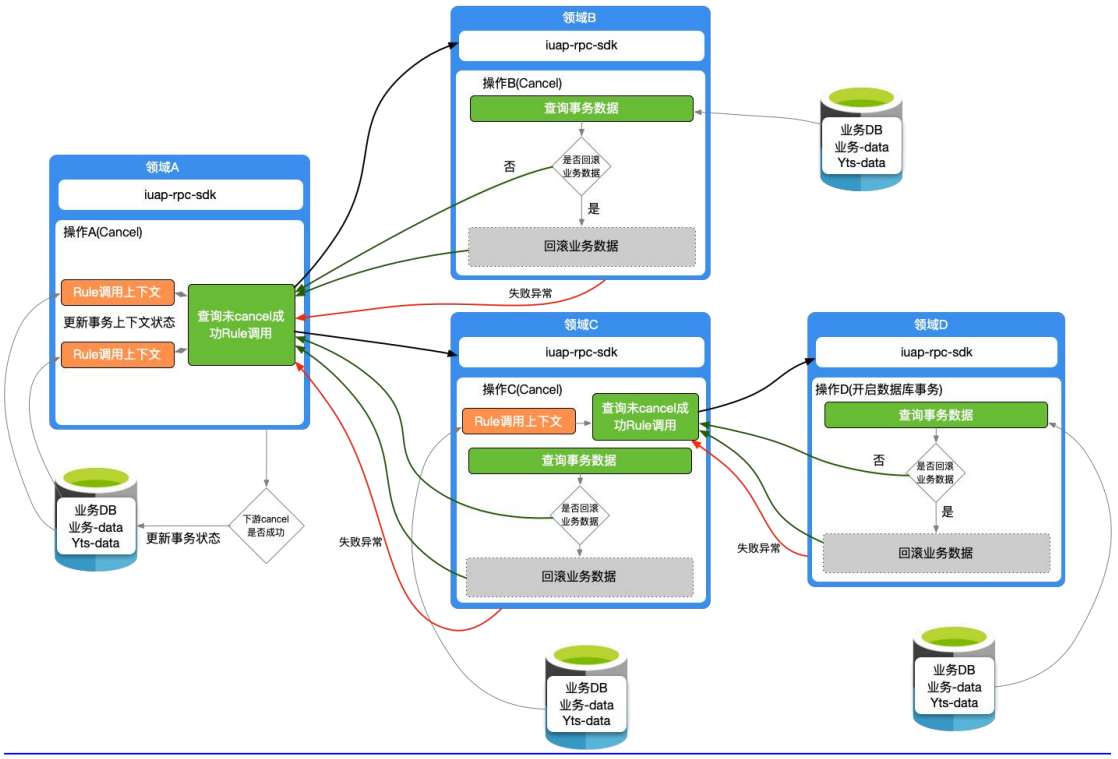
当开发人员从云端查询到回滚异常的事务，并根据系统提供异常堆栈，调用链信息来排查回滚错误原因，排除故障或者数据异常后，点击重试按钮下发事务重试的请求。定时任务收到下发的重试请求后，再次发送事务的回滚操作，回滚没有

正确回滚的事务节点，当所有子事务都回滚成功后，事务状态修改为回滚成功(CANCEL_SUCCESS),否则状态为 CANCE_ERROR ,并再次上报错误的事务数据到云端。

2.3.3.5 分支事务嵌套链式长事务的协调执行



2.3.3.6 分支事务嵌套链式长事务的逐级回滚

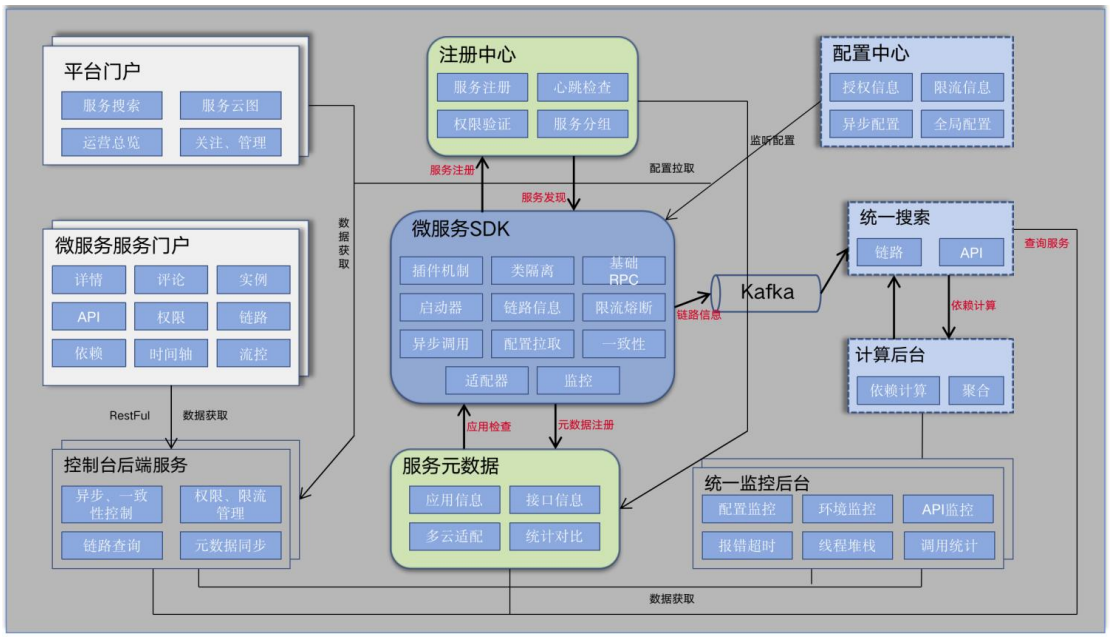


2.3.4 服务/组件依赖

YTS 以 Java 组件包的方式，被业务工程依赖使用，同时配合部分后端支撑服务进行数据的收集和管理。Java 组件层面，底层依赖 IRIS RPC 框架和 MDD 下的规则引擎，依赖 Spring 对于本地事务的管理级平台的持久化框架；后端支撑服务方面，依赖注册中心、配置中心、事务数据管理控制台等。

2.3.4.1 IRIS RPC 调用框架

IRIS 的 RPC 框架负责 MDD 框架的远程规则(Rule)底层调用，或者 IRIS 框架的远程调用执行。涉及到跨服务的 RPC 调用或者是跨服务的远程 Rule 调用的时候，底层的通信机制都是通过 RPC 框架执行的，RPC 框架本身提供了基础的 SDK 和插件机制，为上层的扩展开发提供基础。



2.3.4.2 持久化框架

MDD 框架和 UCF 框架，规范了业务工程的持久化的过程，YTS 框架的一致性保证，源于对全局事务分段方式的数据库本地事务的控制保证。yts java 组件在本身的事务日志持久化的过程中，使用到了业务工程的数据源及事务管理器。本身的数据持久化，采用 Spring JDBCTemplate 来进行数据库的操作。有业务注入 jdbcTemplate 的对象,框架使用该对象进行数据库的操作。业务注入的 jdbcTemplate 的数据源需要和业务数据数据源一致。

2.3.4.3 配置中心

YTS 的动态配置使用 iuap 的配置中心实现，当有配置改动时由配置中心主动通知 YTS 配置组件重新进行加载，实现配置的动态化。

加入到配置管理的 YTS 的配置项包括，是否开启上报事务日志数据、定时调度任务的线程池配置信息、异常模拟的配置信息等等。

2.3.4.4 MDD 规则执行引擎

MDD 框架执行引擎的引用 mdd-rule-common(高版本 mdd-3.0.4 为 mdd-rule-sdk),mdd

-ext-dubbo-query 的组件，并在该组件中添加 YTS 分布式事务的逻辑控制，在跨域服务调用的规则上，可以进行加入分布式事务的选项配置，在调用远程规

则前，YTS 框架根据上下文信息，开启或者加入全局事务，记录事务日志信息，如果跨域规则调用失败，YTS 框架根据条件，进行错误事务的补偿逻辑调用。

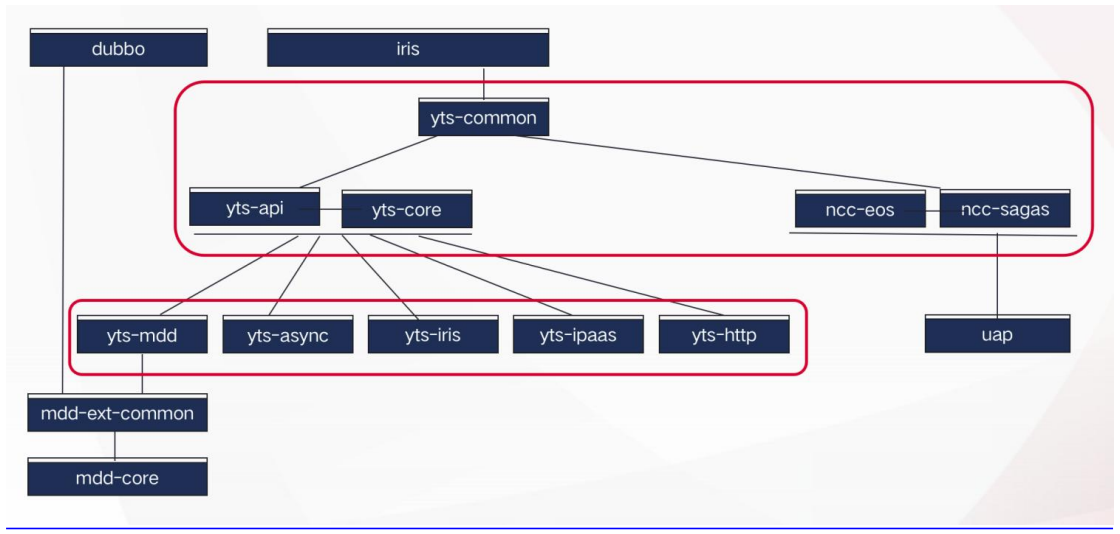
2.3.4.5 Auth-SDK 加签调用

事务数据上报云端以及云端下发重试指令时均需业务服务加签调用来校验该业务是否拥有访问云端的权限，并根据加签的数据来区分上报的数据。

2.3.5 模块/组件拆分

YTS 框架依赖 IRIS，在 YTS 框架的底层，抽象出 yts-common 通用组件，统一定义一致性框架中的枚举、模型、通用接口等，iuap-Sagas 框架和 yts 的各个模块组件共同依赖。

yts 组件本身抽象出 yts-api 和 yts-core，分别定义对外和公共的接口级核心内部控制逻辑。再一层，yts 组件按照运行模型和运行时支持进行划分，分别包括 yts-mdd、yts-iris、yts-http、yts-http-springboot-support,yts-ipaas 等组件包，还包含针对异步机制下的 yts-asyncsagas 组件。



2.3.5.1 yts-core

提供 YTS 分布式事务事务协调器，事务管理器、定时任务以及数据适配等核心功能。

middleware.proteus.config	动态配置变化监听器
yts.compress	rpc 上下文压缩相关功能
yts.config	yts 配置相关的类
yts.dialect	数据库适配
yts.scheduler	定时任务
yts.service.impl	数据库操作以及通用的服务实现
yts.spi	rpc 上下文处理接口
yts.trans	yts 事务协调器以及事务管理器实现
yts.util	常用工具类
yts.*	注入条件匹配，夸线程的线程上下文处理，事务常用工具等

2.3.5.2 yts-iris

该组件是 YTS 事务框架在基于 iris 的微服务的适配层，通过在业务接口设置 @YtsTransactional 将当前业务或开启或加入 YTS 分布式事务并实现事务上下文的传递

删除[华新]：获

服务端调用下游接口，YtsTransactionBeforeInvoke 插件根据业务接口是否有 @YtsTransactional 注解，有注解，则需要将当前事务上下文通过 IRIS 框架传递给服务端，同时，记录事务调用关系，如果没有，则不传递上下文，事务在此分界

服务端接收到请求后，YtsTransactionBeforeExecute 插件根据业务接口是否有 @YtsTransactional 注解及上下文来判断是否开启事务或加入事务

2.3.5.3 yts-http-support

yts-http-support 模块主要实现以下功能：

a. 在业务领域的服务访问云端服务时，对业务领域服务的访问权限进行校验。该模块负责拦截 HTTP 请求远程服务请求，并将 HTTP 的远程请求上下文组装为

Invocation 的对象，进行持久化并将 confirm，cancel 操作相关的信息保存到 TransactionRel 对象中。

b. 将 YTS 事务的上下文（TransactionContext）注入到 HTTP 的请求中传输到被调用的 HTTP 服务端

c. 当 HTTP 服务被执行时拦截请求，根据传入的参数来协调执行 Execute，Confirm，Cancel 等操作，并进行并发以及幂等控制。

d. HttpExecutor 的实现 Sagas，TCC 模式的 Executor 来时间由 TransactionRel 对象，组装为 HTTP(Confirm,Cancel) 请求参数，并执行 HTTP 请求,交由 YTS 事务协调器进行调用。

2.3.5.4 yts-http-springboot-support

该模块主要提供作为 http 事务一致性服务提供方进行请求拦截以及事务管理等功能。目的为简化 http 服务支持 yts 分布式事务一致性的开发工作。

2.3.5.5 yts-asyncsagas

异步消息最终一致性的 SDK 提供服务异步发送消息，被调用的服务订阅消息，通过可靠消息的方式来保证业务数据在微服务之间的可靠性传递，通过可靠消息的传递无需依赖其他服务返回结果的业务场景解决业务数据一致性问题。

异步消息保证消息发送端发送消息，订阅端一定能够消费到消息的机制，避免由于网络异常，服务异常而导致发送的消息不能保证送达到订阅端的问题。

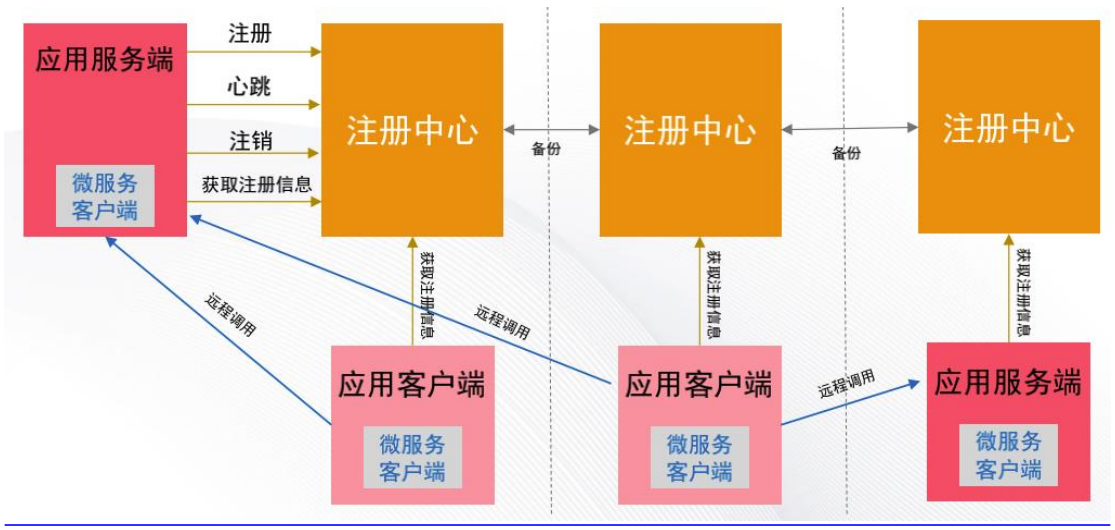
服务端调用下游接口，YtsAsynSagasBeforeInvoke 插件根据业务接口是否有 @YtsTransactional 注解，有注解，则需要将当前事务上下文通过 IRIS 框架传递给服务端，同时，记录事务调用关系，如果没有，则不传递上下文，事务在此分界

服务端接收到请求后，YtsAsyncSagasBeforeExecute 插件根据业务接口是否有 @YtsTransactional 注解及上下文来判断是否开启事务或加入事务

2.3.6 微服务描述

2.3.6.1 注册中心服务

iuap 技术中台默认提供注册中心，供 YTS 框架的 RPC 服务注册和发现服务使用，YonBIP 及各个领域云的微服务，以微服务的形式注册到注册中心，在注册中心可以查看微服务的基础信息和注册信息。



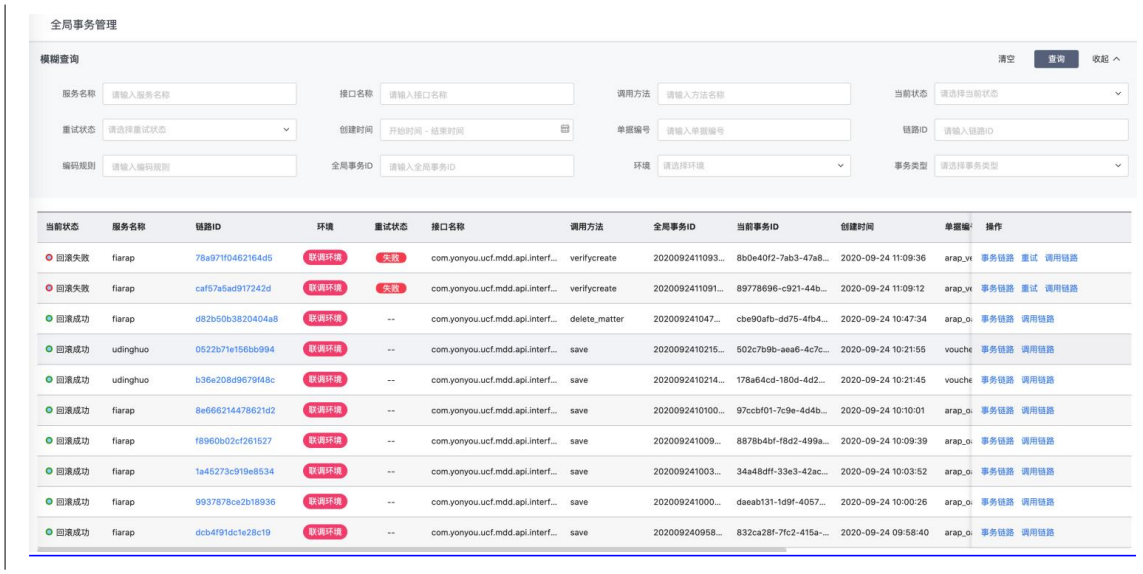
2.3.6.2 配置中心服务

配置中心服务统一存储各个微服务对应的配置文件，其中包含对 YTS 组件运行态造成控制的开关、模拟异常属性等信息。

RPC 框架的 SDK 中包含了配置中心的 SDK，实时接收配置中心管控端可视化的文件编辑的变化。YTS 的对应监听可以接收到配置中心 SDK 的文件变化通知，实时控制业务的全局事务的运行逻辑。

2.3.6.3 eos-console 服务

eos-console 服务为可靠消息和事务日志管理的后端对应工程，接收异常的消息和全局事务信息的上报并进行集中式存储，前端界面展示的全局事务列表信息的数据，来源于本工程的 Rest 服务。

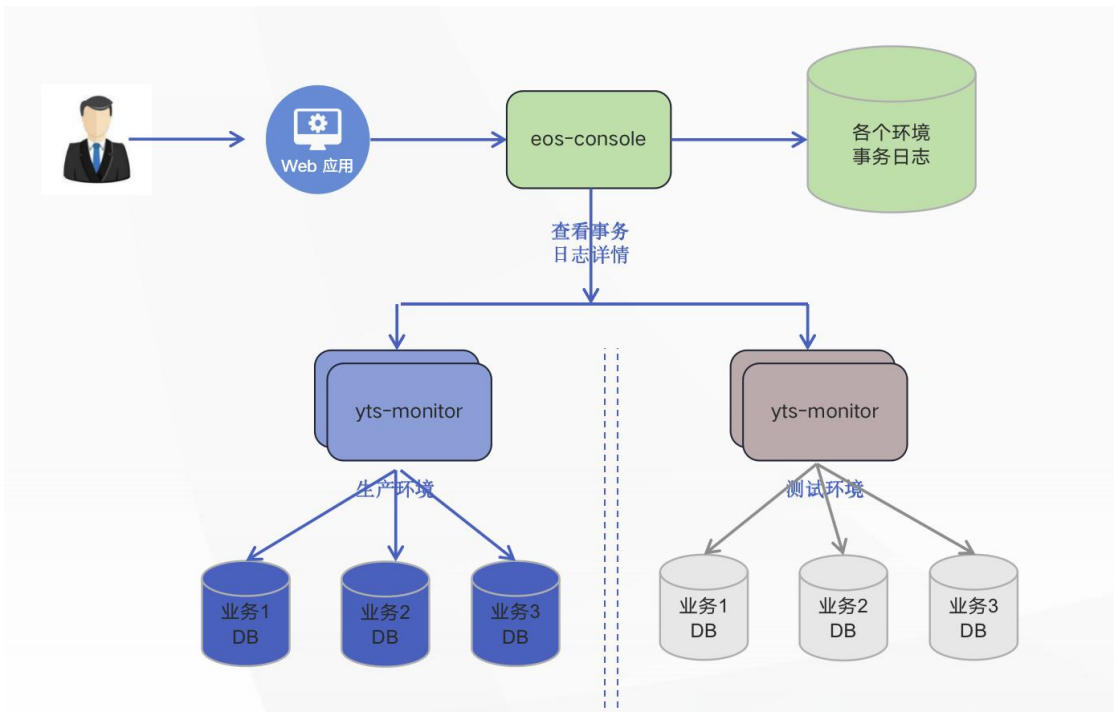


2.3.6.4 yts-monitor 服务

yts-monitor 为 YTS 框架的监控后端服务，在展示错误事务数据列表时，前端请求发送到 eos-console 获取主数据，通过主数据的全局事务 ID，可以向 yts-monitor 发起数据查询请求，获取本全局事务的详细信息。

技术平台下的微服务部署，是按照环境进行隔离的，针对不同的环境，有数据及网络的隔离，不同的环境，yts-monitor 各部署一份，在工程中配置不同的业务库的数据源，根据全局事务 ID，到不同的业务库中抓取事务数据详细信息。

同时 yts-monitor 启动了定时任务，定时的扫描业务库中过期的数据，进行清理。



2.4 开发架构设计

删除[华新]:

2.4.1 依赖、适配中间件

删除[华新]:

中间件	版本	作用
MySql	5.6+	DB
Redis	3.2.8+	Sagas 上下文、MQ 消息、ack 状态等

设置格式[华新]: 正文

2.4.2 依赖的核心技术栈和框架

删除[华新]:

框架	版本	作用
Java	1.8	SDK 及后端工程

Spring	4.0+	后端 java 工程引用 Spring 框架
Maven	3.2+	工程构建
Git	1.8	代码管理
JWT	3.1.1	加签验签
HttpClient	4.5.3	Restful 服务调用
OKHttp	4.0.1	RPC SDK HTTP 调用

2.4.3 配置说明

2.4.3.1 YTS 引入及配置

MDD 框架需要支持 YTS 分布式事务，则需要添加依赖：

```
<dependency>
    <groupId>com.yonyou.cloud</groupId>
   <artifactId>yts-core</artifactId>
</dependency>
```

Iris 框架需要支持 YTS 分布式事务，则需要添加依赖

```
<dependency>
    <groupId>com.yonyou.cloud</groupId>
   <artifactId>yts-iris</artifactId>
</dependency>
```

在服务配置中增加：

```
<!--在 iris rpc 项目中加上如下 bean 配置-->
<bean id="transactionService"
class="com.yonyou.cloud.yts.service.impl.JdbcTplTransactionService"/>
<bean id="transactionRelService"
class="com.yonyou.cloud.yts.service.impl.JdbcTplTransactionRelService" />
<bean class="com.yonyou.cloud.yts.trans.TransactionSynchronizer" />
```

```
<!--在 mdd 或 iris rpc 项目中都需要加如下配置-->
<bean    id="irisSagasExecutor" class="com.yonyou.cloud.yts.IrisSagasExecutor" />
```

2.4.3.2 基于 MDD 的规则配置

将 Rule 添加 YTS 分布式事务，需要在 BillRuleRegister 数据库 Rule 的配置中增加：

```
{
  "needDTC": "true",
  "transactionType": "sagas",
  "params":"param,testcacel"
}
```

其中 params 的值为在执行 cancel 操作时，需要从正向调用的参数中或者那些参数的值来调用 cancel 方法，如果需要多个 key 则以半角“,”分割。

2.4.3.3 YTS 动态配置

在开发者平台配置中心增加 mwclient.json 的配置文件，配置文件采用 json 格式进行配置，yts 的配置的键为"yts"，内容为 json 格式。主要有以下几个配置项：

taskEnable

yts 定时任务是否启用，取值为 true,false,默认为 false 状态，即不启用定时任务。

taskConfig

为定时任务的配置，键为定时任务名称，值为定时任务的表达式。键的取值有：

- insertTransToCloud 上报错误事务数据以及上报状态更新的事件
- pullTransToLocal 从分布式事务云管理中心下载需要重试的任务
- doRetryingTask 扫描在云端设置重试状态的事务，进行重试操作
- doTransHangTask 处理由于服务异常导致挂起分布式事务
- doFrozenUpTask 处理数据隔离时冻结数据的解冻

值为定时的表达式，可以使用 spring schedule 兼容的表达式，如 "5 * * * * *",如果表达式只有一个数字，则为固定指定毫秒执行的定时任务，如果值为 "5000",则该任务为上次执行完 5 秒后再次执行。任务如果不进行设置，则采用默认的间隔时间。每个任务的默认间隔时间为：

- insertTransToCloud 默认间隔时间 15 秒(15000)
- pullTransToLocal 默认间隔时间 15 秒(15000)
- doRetryingTask 默认间隔时间 15 秒(15000)
- doTransHangTask 默认间隔时间 30 秒(30000)
- doFrozenUpTask 默认间隔时间 15 秒(15000)

任务配置后及时生效无需重启服务。

compressor

事务上下文压缩的配置，由于事务上下文可能体积较大，对数据库的 IO 影响较大，可以在保存时进行压缩，减少数据的体积。压缩的主要配置为采用的压缩器的实现，以及压缩器的配置参数。现在压缩器只有一个"FlateCompressor"使用 java 的 Deflater 压缩,Inflater 解压缩，该压缩算法可以在压缩速度和压缩体积上做调整，级别从 0-9，0 为不压缩，数字越大，压缩体积越小，但是压缩的速度越慢。如果级别不设置，则使用默认 1 的 level 进行压缩。配置示例如下：

```
{
    "name":"FlateCompressor",
    "config":{
        "level":1
    }
}
```

enableDTC

iris 框架是否开启分布式事务支持，开启分布式支持需要 iris 接口上添加"YtsTransactional" 注解，如果没有"YtsTransactional" 注解，打开分布式的开关也不能开启分布式事务。 取值为： true 或者 false

disableDTCList

分布式事务禁用接口或者方法列表，当"enableDTC"为 true 时，如果需要关闭某些接口后者接口的某个方法，可以在该配置项下配置需要禁用的接口，或者是接口的 method 名称，method 的名称需要有参数类型的签名，如：

```
[

"com.yonyou.iuap.rpc.service1.IFastJsonTestServicefastJsonTest.fastJsonTest(java.lang.String,java.lang.String,java.util.Map)",
]
```

如果想禁用整个接口可以只填写接口的全名即可：

```
[
    "com.yonyou.iuap.rpc.service1.IFastJsonTestServicefastJsonTest"
]
```

2.4.3.4 本地事务的 AOP 配置

2.4.3.4.1 MDD 框架 AOP 配置

找到 config 项目下 spring-com/applicationContext-db.xml

在 <tx:attributes> 下增加如下两个配置

<tx:method name="doAction" propagation="REQUIRED"/>

<tx:method name="persist" propagation="REQUIRES_NEW"

rollback-for="java.lang.Exception"/>

在 <aop:config>项 的 expression 表达式中增加 yts 的事务配置：

execution(* com.yonyou.cloud.yts.service..*.*(..)) || execution(*

com.yonyou.ucf.mdd.ext.service..*.*(..))

2.4.3.5 iris 和 mdd 同时使用的配置

2.4.3.4.2 IRIS 框架 AOP 配置

IRIS 框架的 AOP 配置在依赖的 yts-iris 的包中已经添加了 AOP 相关的配置，业务服务无需单独进行添加

2.4.4 YTS 事务接口开发

2.4.4.1 基于 YTS 的 Sagas Rule 开发

MDD 框架 Sagas，TCC 模式采用业务实现接口的方式接口的定义：

Sagas 模式的接口

```
public interface ISagaRule {
   RuleExecuteResult cancel(BillContext billContext, Map<String,Object> paramMap)
throws Exception;
```

}

2.4.4.2 基于 YTS 的 TCC Rule 开发

```
public interface ITccRule {
   RuleExecuteResult confirm(BillContext billContext, Map<String,Object> paramMap)
throws Exception;
   RuleExecuteResult cancel(BillContext billContext, Map<String,Object> paramMap)
throws Exception;
}
```

2.4.4.3 MDD 框架 sagas 模式通用的规则(backWriteSaveRule)

MDD 框架的推拉单规则（Rule）为统一的规则，无需业务开发实现，业务领域如果有推拉单的服务，只需引入 mdd-ext-common 后，在 billruleregister 表的 config 字段添加：

```
{
"needDTC": "true",
"transactionType": "saga",
"params":"param"
}
```

的配置即可。

2.4.4.4 基于 YTS iris 微服务 sagas 模式开发

```
/** 业务方法 */
@YtsTransactional(mode = "sagas", cancel="cancelUserAdd")
userAdd(UserInfo userinfo);
/** 业务方法的回滚方法 */
cancelUserAdd(userInfo);
```

2.4.4.5 基于 YTS iris 微服务 tcc 模式开发

```
/** 业务方法 */
@YtsTransactional(mode = "tcc", cancel="cancelUserAdd", confirm="confirmUserAdd")
userAdd(UserInfo userinfo);
/** 业务方法的资源锁定方法 */
confirmUserAdd(userInfo);
/** 业务方法的回滚方法 */
cancelUserAdd(userInfo);
```

2.4.4.6 HTTP 服务端 Sagas 模式开发

```
@GetMapping("/save")
@YtsTransactional(mode = TransactionMode.SAGAS, cancel="cancelSagas")
public String sagas(HttpServletRequest req, @RequestParam("uid") Long uid, @RequestParam("amount") Long amount) throws SQLException {

        // 正向逻辑

    return "hello";
}

public String CancelSagas(HttpServletRequest req, @RequestParam("uid") Long uid, @RequestParam("amount") Long amount) throws SQLException {

        // 回滚逻辑

    return "hello";
}
```

调用支持 YTS 分布式事务 HTTP 的代码：

注入 YtsHttpClient 的服务

使用如下方式调用：

```
String url = "http://demo.app.yyuap.com/yts/save?uid=" + uid;
String result = ytsHttpClient.get(url);
```

2.4.5 异常模拟与测试验证

为了方便测试人员模拟分布式事务异常的场景，框架在 mdd 框架以及 iris 框架的基础上进行了扩展，可以在远程 Rule 调用或者远程服务调用时根据配置在指定的位置抛出相应的异常。

异常的数据定义（MockException）：

属性	类型	说明
position	string	位置放在操作前还是操作后，取值 front 或者 rear
type	string	异常类型，取值 crash 服务宕机,sql 数据库保存失败,timeout 调用超时
invokePosition	string	模拟异常执行的位置，取值 CALLSERVICE(服务调用 端),INVOKE_SERVICE(服务执行端)
msg	string	异常描述信息
action	string	模拟异常在 mdd 框架 rule 的 action 操作，默认为 execute，回滚时为 cancel
timeout	int	超时(timeout)异常时,超时时间的设置单位为秒，超时的异常在运行期不会直接抛出异常。超时异常的操作是让运行该异常的线程直接休眠相应秒数，如果是 RPC 的被调用端如果 rpc 超过了上游设置的超时时间则会导致上游抛出超时异常，来模拟网络抖动或者 RPC 服务无法访问的场景

配置时以 json 字符串的方式进行配置，样例：

```
{
   "type":"sql",
    "msg":"sql 执行异常",
    "invokePosition":"CALL_SERVICE",
    "position":"REAR"
}
```

2.4.5.1 MDD 框架使用元数据方式模拟异常

在 MDD 框架的 BillRule 的原数据中，增加 mock 的属性，用来记录该 Rule 是否有模拟异常的设置。当调用或者执行该 Rule 时，根据 mock 字段的值解析为 MockException 的数据，根据设定位置抛出异常。

2.4.5.2 MDD 框架使用动态配置方式模拟异常

在开发者中心的配置管理选择需要配置模拟异常的服务。

（https://developer.yonyoucloud.com/#/）

添加 /mwclient.json 的配置文件配置文件的内容为 JSON 格式的字符串，mock 异常的设置的 key 为"yts.mock"

配置示例：

正向操作异常：

```
{
   "yts.mock": {
       "lc2h5i1r_aa_warehouse_save_sagsServiceBRule": {
           "type": "sql",
           "msg": "sql 执行异常",
           "invokePosition": "INVOKE_SERVICE"
       }
   }
}
```

回滚操作异常：

```
{
   "yts.mock": {
       "lc2h5i1r_aa_warehouse_save_sagsServiceBRule_cancel": {
           "type": "sql",
           "msg": "sql 执行异常",
           "invokePosition": "INVOKE_SERVICE"
       }
   }
}
```

yts.mock 的 key 为： 租户标记+ "_" + 单据类型 + "_" + 动作 + "_" + ruleId + ["_" + action](如果为正向操作则不填写)，值为 MockException 的对象。

2.4.5.3 iris 框架的异常模拟

iris 配置项采用 JSON 结构进行存储，key 为"接口名" + "." + "方法名" + "(" + 方法入参类型签名 + ")", "方法入参类型签名"为入参的类型 Class 的名称如果入参为多个则使用","进行分割。iris 在 mwclient.json 中的配置样例：

```
{
  "yts.mock":{
"com.yonyou.iuap.rpc.IUserService.UpdateProfile(java.lang.String,java.lang.String,java.util.Map)": {
            "type": "sql",
            "msg": "sql 执行异常",
            "invokePosition": "INVOKE_SERVICE"
        }
  }
}
```

运行时根据调用的信息匹配 Mock 异常信息，如果能够匹配到则将 Mock 的字符串解析 MockException 的数据，然后根据设定的位置抛出异常或者让线程休眠指定的时间。

2.5 运行架构设计

2.5.1 异常处理设计

2.5.1.1 事务异常点分析

1) 非根节点业务异常，本地事务提交，调用方接收到异常，全局事务失败，事务回滚，需要调用当前节点回滚方法

2) 非根节点业务异常，本地事务回滚，调用方接收到异常，全局事务失败，事务回滚，当前节点不需要调用回滚方法

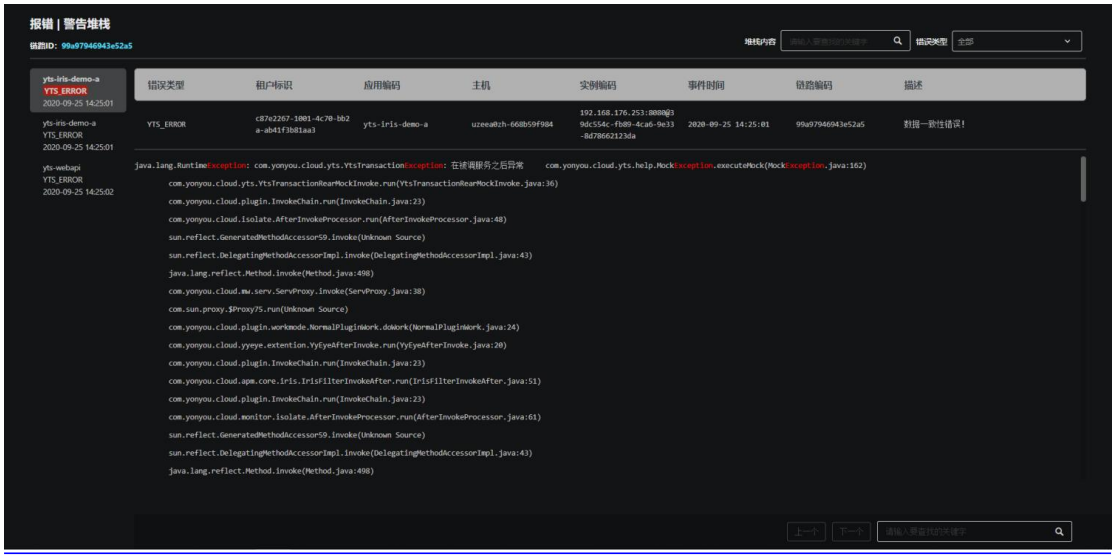
3)根节点业务异常，本地事务已提交，全局事务提交，但会返回前端异常信息

4)根节点业务异常，本地事务回滚，回滚全局事务，前端会提示异常信息

5)事务协调器异常，事务状态导致导致未能 Confirm/Cancel，并且根节点事务状态未能修改成功，定时器处理长时间 Confirming 状态的事务

2.5.1.2 错误事务信息上报

YTS 框架事务异常信息通过 rpc 框架链路功能模块统一上报收集到云端，并将事务相关的信息标识为 YTS_ERROR 类型，当全局事务异常后，通过该全局事务的 trace_id 获取全局事务的异常信息，这样就可以快速定位到导致事务异常的原因



YTS 事务异常日志

2.5.1.3 错误事务重试

如果分布式事务出现回滚失败，则事务管理中心则会出现事务的数据，在事务数据的操作中会出现重试的按钮，如果开发人员根据错误堆栈信息确认程序已经解决了出现回滚错误的逻辑，则可以点击重试按钮。点击重试按钮后 YTS 的 sdk 将会将重试的任务下载到本地并有重试的定时任务进行重试。重试状态变更后，由上报的定时任务将重试的状态上报到事务管理重新。

2.5.2 定时任务及线程池设计

2.5.2.1 错误数据上报定时任务

错误数据上报定时任务定时扫描数据中状态为非正常执行的事务信息（包含回滚成功的事务）。将错误的事务数据，上报到异常事务处理中心。

2.5.2.2 错误堆栈上报

当服务运行时出现异常，则程序捕获该异常并通过 yts-SDK 将错误的堆栈上报到对应的微服务中，方便开发人员查询，定位问题。

2.5.2.3 回滚动作异步线程池

当程序调用远程的服务出现异常或者应用本身出现异常，导致事务不能正常提交。则需要在事务结束后将事务的信息交由回滚处理的线程池进行事务的回滚操作。为了方便追踪问题，该线程池执行时会复制事务线程的线程上下文，将事务信息，链路信息在 RPC 调用时进行正常的复制，方便链路追踪以及问题排查。

2.5.2.4 挂起事务的处理

由于服务运行异常导致服务异常退出而没有办法进行分布式事务的后续处理，导致分布式事务不能正常完成。该定时任务扫描 5 分钟前还没有完成的事务数据，将该事务进行回滚处理，避免由于服务异常退出导致的分布式事务挂起不能正常结束导致的数据不一致问题。

2.5.3 可靠性设计

为了减少分布式事务返回结果的幂等性，在 RPC 调用是禁用 RPC 调用的重试功能，避免不必要的幂等性控制减少业务处理没有幂等控制造成数据不一致的问题。

2.5.3.1 分布式事务的 TC，RM 均采用分布式的架构

为了减少单点故障，单点性能瓶颈以及 RPC 通讯的次数，TC，RM 均采用分布式架构，日志和事务控制均在业务节点进行存储和控制，采用级联向下调用的机制，而不是采用的统一中心进行控制的方式。该方式第一减少了单点故障，而且

在大多数事务不会失败的情况下，在事务无需回滚操作时可以不进行事务控制的 RPC 调用。

2.5.3.2 分布式事务的日志存储和业务数据库在同一个数据库

为了分布式事务日志和的业务操作的的一致性，框架将该节点设计到底分布式事务的日志和业务数据库放在同一数据库，避免由于跨库操作导致的业务数据和分布式事务日志数据不一致的问题。分布式日志存储和业务操作使用成熟的数据库的事务控制，来实现数据的一致性保证。

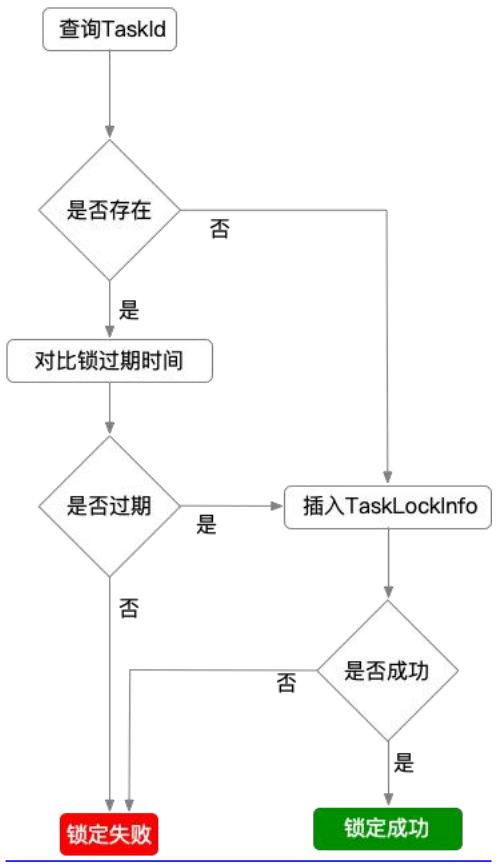
2.5.4.3 定时任务处理服务异常导致的分布式事务没有完成的问题

使用定时任务定时扫描由于服务异常没有能够完成的分布式事务，由于分布式日志有些操作使用新开事务进行存储的机制，当服务异常分布式事务不能正常完成时分布式的日志已经进行了保存，通过定时任务扫描数据库的方式将异常的日志扫描出来，然后进行回滚操作，完成异常的分布式事务

2.5.3.4 定时任务加锁控制

由于服务部署时均为多节点部署，为了防止多个服务节点同时处理同样的任务，所以定时任务运行时采用锁的机制，使同一任务只会在一个节点执行。为了减少其他中间件的引入，定时任务的锁采用 mysql 数据库结合算法来实现。

Lock 的操作逻辑：



Unlock 逻辑：

锁定时会返回 LockId,unlock 是删除值为 LockId 的记录即可。

- 如果由于服务异常等原因没有执行 unock 操作，则可能会产生一条脏数据，不影响锁的逻辑。

2.5.3.5 幂等、并发控制

2.5.3.5.1 正向业务逻辑幂等

mdd 或 iris 框架为了容错，通常在调用远程服务异常时，会重试调用以提高正确率，这就容易导致问题，如果业务本身没有做幂等性设计，那么，业务方法将会重复执行，这会导致严重的问题，比如，如果业务是给客户打款，重复执行将会给客户打款多次，造成严重损失，是绝对不可以接受的严重 bug，针对此问题，yts 分布式事务一致性框架提供了幂等处理逻辑，保证正向业务不会重复执行。

yts 幂等方案，yts 框架保证即使不做幂等，也不会出现幂等问题。在 yts 框架参与的分布式调用，针对每次调用，由调用方生成了一个本次调用的 TXID，标识本次调用,服务端执行业务方法之前先插入本次事务 LOG 表(YTS_TRANSACTION)，

初始状态为 CONFIRMING，其中的 tx_id 字段为唯一索引，因此，服务端如果已经执行过了，再次执行时会被阻止。

编码规范：上述幂等控制是基于框架自身的请求重放的重试，而不是业务开发人员获取异常后重新调用请求，非框架自身的重试是两个不同的请求，其 TXID 是不一致的，服务端就无法幂等。

2.5.3.5.2 回滚逻辑幂等

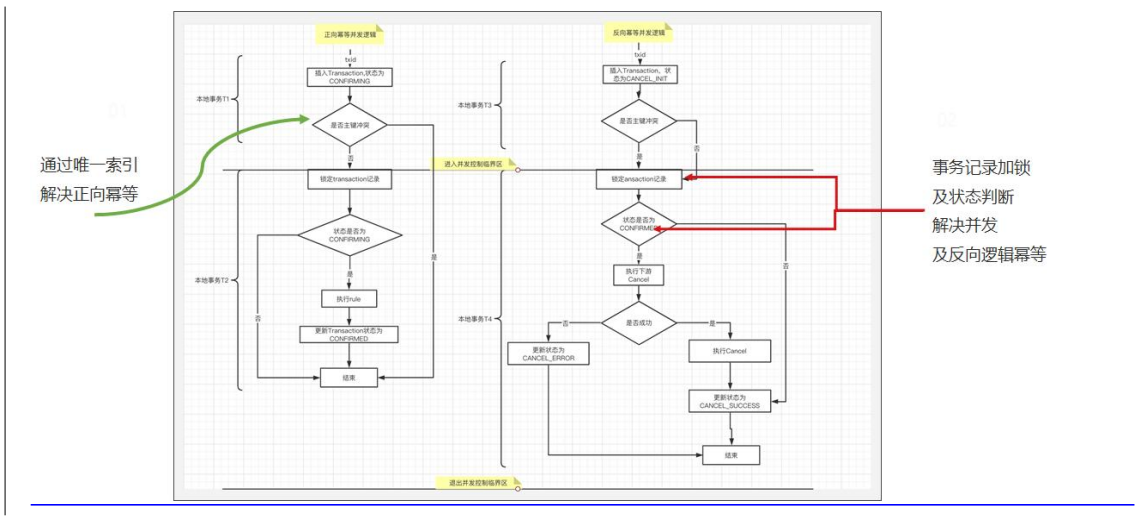
事务回滚时也可能出现重试的情况，因此，回滚时也要考虑幂等的情况发生，方案同正向业务逻辑幂等一致，不管是正向业务还是回滚，都需要往事务 LOG 表插入此次分布式事务记录，如果是回滚插入，其初始状态为 CANCEL_INIT，同一个事务的正反向 TXID 相同，因此，反向先执行会阻止正向滞后执行的情况发生。

2.5.3.5.3 并发控制

yts 并发问题是指如果正向执行过程中，因为超时等原因造成调用端发起回滚操作，此时正向数据已经执行但并未提交，反向没有执行具体得回滚业务数据的逻辑，当正向执行完毕时，提交了业务数据，此时事务已经回滚，导致数据不一致的情况

yts 解决方案：基于幂等逻辑，正向执行时插入本次分布式事务 LOG，执行业务之前立即锁定该记录，在业务执行期间持有锁，直到业务提交，如果期间执行回滚，也需要锁定该记录，显然，此时锁由正向业务持有，回滚逻辑会等待，直到正向释放，正向时候后提交业务数据，接着反向拿到锁继续执行，回滚正向提交的业务数据。期间如果反向超时，即使正确回滚了，根节点的状态也是 CANCEL_ERROR，会上报到事务控制台。此时，事务控制台需要重试完成事务 LOG 的一致性。

并发幂等控制流程图



2.5.3.6 隔离性机制

YTS 目前提供基于冻结字段的半自动隔离机制，即启用隔离需要显示调用，解除隔离由框架自动进行，如上图所示，开启事务后，业务本地事务提交前，会修改需要冻结的单据，如果修改成功，则当前事务冻结成功，否则当前事务会抛出异常，本地事务回滚，全局事务失败，全局事务失败后

解冻时机：Sagas 模式：全局事务异常时，回滚成功时参与者解冻，全局事务成功时，定时器根据该全局事务根节点事务状态决定是否解除冻结

TCC 模式或异步 Sagas 模式：第二阶段 Confirm/Cancel 成功时解除冻结

可靠消息模式：不支持隔离，不支持全局事务，只提供努力向前机制，没有回滚或提交机制

2.5.4 高性能设计

2.5.4.1 异步数据上报

异常数据上报，以及错误堆栈上午均采用异步的方式尽量减小对应用的影响

2.5.4.2 数据库数据清理及索引调优

根据 YTS 运行的情况，合理的建立数据库索引，详见: 2.6 数据库设计。

2.5.5 安全性设计

2.5.5.1 加签验签

数据上报下载均使用 auth-sdk 进行加签以及延签的认证机制。方式应用程序访问到其他应用的数据。

2.5.5.2 注册中心逻辑环境隔离

注册中心通过逻辑环境隔离使不同环境的应用进行隔离，防止不同环境的应用调用到不同环境服务的问题。

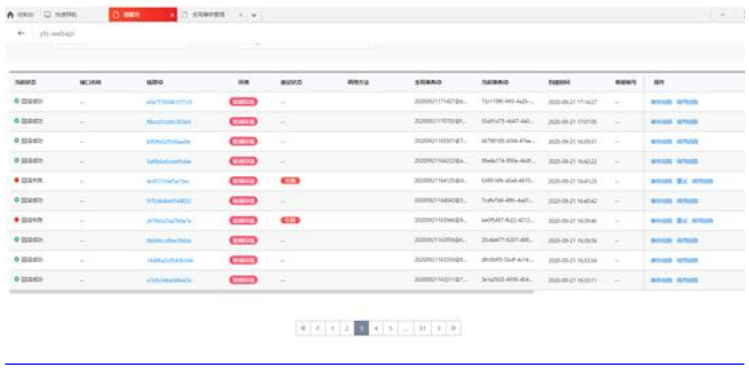
2.5.5.3 监控数据环境隔离

监控数据也通过环境隔离，不同环境的开发人员只能看到对应环境的数据，防止开发人员存在越权的风险。

2.5.6 可维护性设计

2.5.6.1 问题排查

YTS 分布式事务框架提供了完善的容错功能，通过与微服务治理平台整合，可以非常方便的排查及定位事务一致性问题，同时也提供了友好的交互界面对不一致的全局事务进行处理全局事务异常回滚后，YTS 的 SDK 会将回滚或失败的全局事务信息上报到控制台，微服务有权限的用户可以在开发者中心的事务管理页签看到回滚的事务信息



YTS 异常事务列表

在该列表中，可以点击链路 ID 查看具体的导致事务回滚的异常信息，帮助开发定位问题，回滚成功的全局事务，各个参与者数据都已正确回滚。如果回滚或提交失败，则可以点击右边的重试按钮，直到成功为止，达到一致。

删除[华新]：

设置格式[华新]：缩进：首行缩进：7.4 毫米，行距：单倍行距，正文文本

2.5.6.2 YTS 分布式事务时间报告

通过 hubble 链路跟踪，可以查看事务执行过程中的事件执行报告，根据报告，可以分析分布式事务执行到了哪一步，配合链路日志，查看出错原因

Hubble 分布式事务报告

删除[华新]：

设置格式[华新]：居中，行距：单倍行距，正文文本

2.5.6.3 动态参数配置（mwclient.json）

为了在运行时不停机进行参数的调整以及方便私有云部署时方便对组件功能进行打开和关闭一些功能，YTS 的定时任务启动，定时任务间隔时间，事务上下文是否启动压缩，分布式事务功能开关以及为了方便测试，对分布式事务的各环节进行模拟异常的功能的参数配置均采用动态的参数配置。可在开发者中心（YonBuilder）的配置中心进行配置，配置无需重启服务即刻生效，方便运行期对参数进行调整，并提高服务的的可用性。

具体的配置参考 2.4.3.3 YTS 动态配置

2.5.6.4 自动清理过期事务日志数据

根据配置参数中设置的清理任务启动时间，分布式日志保留的天数，以及可清理状态，以及批量清理之间的间隔每天进行分布式日志的循环（删除一批休眠一定的毫秒数）小批量删除，避免一次删除占用大量的数据库的 IO。

自动清理的配置参数如下：

cleanTaskCron：　　定时任务启动的表达式

logRetainDays：　　保留日志的天数默认为 7 天

ytsBatchDeleteCount：每次删除的记录数

ytsBatchDeleteSleepMillSeconds：　删除一次休眠的毫秒数，防止一直循环执行占用大量的数据 IO

2.6 数据架构设计

iuap YTS 框架使用分段的本地事务、定时任务等方式保证长活事务的一致性。YTS java SDK 部分及 auth-sdk HTTP client 后续支持多种数据库类型，目前支持 mysql 数据库。

2.6.1 关系数据库存储设计

iuap YTS 框架使用关系型数据库存储对应的事务日志和关系，共包含 4 张表。

表名	含义	是否按天分表	是否包含序列
yts_transaction	yts 事务	否	否
yts_transaction_rel	yts 事务关系	否	否
yts_task_lock	yts 分布式锁	否	否
yts_context	yts 事务冻结、事务上下文信息	否	否
yts_biz_binds	yts 单据与事务 LOG 关系表	否	否

2.6.1.1 yts_transaction(yts 事务表)

字段名	含义	类型	MySQL 类型
id	分布式事务的唯一标识	字符串	varchar
status	事务执行状态	字符串	varchar
retry_status	重试状态：error、retrySuccess、retryFail	字符串	varchar
provider_id	租户 ID	字符串	varchar
service_name	服务名称	字符串	varchar
env	环境	字符串	varchar
method_name	方法名称	字符串	varchar

gtx_id	全局事务 ID	字符串	varchar
tx_id	事务 ID	字符串	varchar
ptx_id	父事务 ID	字符串	varchar
rule_id	规则 ID	字符串	varchar
trace_id	链路跟踪 ID	字符串	varchar
bill_no	表单编码	字符串	varchar
cancel_method	回滚方法	字符串	varchar
type	事务类型	字符串	varchar
create_time	创建时间	Long	bigint
interface_name	接口名称	字符串	varchar
update_time	更新时间	Long	bigint
confirm_method	正向方法	字符串	varchar
reported	上报状态	字符串	varchar
context	上下文	字符串	varchar
call_tag	调用 tag 标签	字符串	varchar
invocation	服务调用的参数信息	byte[]	longblob
pk	分布式事务日志的主键	Long	bigint
all_confirmed	子操作是否都确认	Integer	int
mode	分布式事务工作模式	字符串	varchar

yts_transaction 表的索引设计：

索引名	包含字段	索引类型
pk	pk	主键索引
idx_update_time_status	update_time,status	
idx_service_name_retry_status	service_name,retry_status	

idx_id	id	
idx_gtx_id	gtx_id	
idx_tx_id	tx_id	唯一索引
idx_reported_service_name	reported,service_name	

2.6.1.2 yts_transaction_rel(yts 事务关系表)

字段名	含义	类型	MySQL 类型
id	事务的唯一标记	字符串	varchar
provider_id	当前服务的商户唯一主键	字符串	varchar
method_name	excute 的方法名	字符串	varchar
retry_status	重试状态：error、retrySuccess、retryFail	字符串	varchar
reported	上报控制台状态	字符串	varchar
rule_id	规则 ID	字符串	varchar
trace_id	链路跟踪 ID	字符串	varchar
bill_no	业务单据号	字符串	varchar
gtx_id	全局事务唯一标记	字符串	varchar
ptx_id	父事务唯一标记	字符串	varchar
tx_id	当前事务唯一标记	字符串	varchar
call_service_name	调用的服务名	字符串	varchar
call_interface_name	调用的接口名称	字符串	varchar
call_method_name	调用的方法名	字符串	varchar
service_name	服务名称	字符串	varchar
interface_name	接口名称	字符串	varchar
cancel_method	回滚的方法名	字符串	varchar
invocation	调用的参数	byte[]	longblob

cancel_invocation	调用 cancel 方法的参数	byte[]	longblob
server_provider_id	服务提供者的商户唯一主键	字符串	varchar
create_time	创建时间	Long	bigint
update_time	更新时间	Long	bigint
confirm_method	confirm 方法名	字符串	varchar
confirm	confirm 状态	Integer	int
env	环境	字符串	varchar
pk	依赖服务的主键	Long	bigint
parent_pk	依赖服务的主事务主键	Long	bigint
call_tag	调用的标签	字符串	varchar
mode	分布式事务工作模式(saga/tcc)	字符串	varchar
retry_count	重试次数	Integer	int
status	子操作的状态	字符串	varchar

yts_transaction_rel 表的索引设计：

索引名	包含字段	索引类型
pk	pk	主键索引
idx_parent_pk	parent_pk	
idx_ptx_id	ptx_id	
idx_gtx_id	gtx_id	
idx_id	id	

2.6.1.3 yts_task_lock(yts 分布式锁表)

字段名	含义	类型	MySQL 类型

pk	主键	Long	bigint
task_id	任务 ID	字符串	varchar
expire_time	过期时间	Long	bigint

2.6.1.4 yts_context(yts 上下文表)

字段名	含义	类型	MySQL 类型
gtx_id	全局事务 ID	字符串	varchar
ptx_id	上级事务 ID	字符串	
tx_id	事务 ID	字符串	varchar
context	上下文内容	大字段	blob
type	/** * 用户自定义上下文信息 */ USERPARAMS("params", "用户自定义上下文信息"), /** * 冻结的表单信息 */ FROZENITEM("frozens", "冻结的表单信息"), /** * 上游设定的 ACK 回调信息 */ UPSTREAMACKCONTEXT("upcontext", "上游设定的 ACK 回调信息"), /** * 非 rpc 服务端开启事务时需要设置的第二阶段回调接口	字符串	varchar

	*/ SENCONDPHASEBALLBACK("secondphasecb", "第二阶段回调上下文"), /** * 下游设定的 ACK 回调信息 */ DOWNSTREAMACKCONTEXT("downcontext", "下游设定的 ACK 回调信息");		
env	环境	字符串	varchar
create_time	记录创建时间	long	bigint
update_time	记录最后更新时间	long	bigint

2.6.1.5 yts_biz_binds(yts 单据与事务 LOG 关系表)

字段名	含义	类型	MySQL 类型
gtx_id	全局事务 ID	字符串	varchar
ptx_id	上级事务 ID	字符串	varchar
tx_id	当前事务 ID	字符串	varchar
biz_table	业务表	字符串	varchar
biz_pk	业务表主键	字符串	varchar
pk_values	主键值	字符串	varchar

设置格式[华新]：正文

2.6.2 多数据库适配

YTS 的 sdk 在运行时根据配置的数据库连接信息获取当前服务使用的数据库类型，并根据数据库类型在数据库适配的配置中查询对应的数据库操作 SQL 来正确处理数据的插入，更新，删除以及相应的数据库锁操作。通过多数据库的适配，支持分布式事务具有在多数据库环境下保证数据一致性的能力。目前已经支持的数据库有 Mysql、SqlServer，达梦

2.7 部署架构设计

后端监控服务部署：