# **技术中台的介绍**

## 鸿鹄平台是什么？

**鸿鹄平台**是我行自主研发的大型分布式基础技术平台。该产品的**目的**是我行要在技术上实现完全自主可控，从开发态，运行态，到运维态三大维度为分布式应用系统提供全方位的技术支持。为我行由传统的“总线架构”向“高可扩展的企业级单元化微服务架构”的转型奠定基础，建立起系统与系统之间的标准化**服务调度**，标准化**通信传输**，标准化**服务治理**的完整生态。

鸿鹄平台目前是构建于腾讯PAAS之上的技术平台。后续逐步扩展为多技术栈兼容（微服务技术体系：腾讯TSF，Spring Cloud，Spring Cloud Alibaba；数据库：MYSQL,TDSQL,**HTAP；**消息队列：TDMQ,CMQ；缓存：腾讯CRS，redis）

要求应用不能越过鸿鹄平台直接腾讯的技术组件；

## 鸿鹄平台的六项核心技术组件

开发态：

1. 分布式应用的开发框架 T-DDF
2. 基础框架（以SDK的形式提供）

* 报文规范
* 全局流水号
* 交易路由
* 微服务治理
* 微服务网关
* 日志（T-POMP）

1. 开发平台

* 联机开发平台

服务编排，脚手架

* 批量开发平台

1. 增强服务

* 分布式事务

鸿鹄平台实现分布式事务有两种方式：编排事务和注解事务。

交易使用流程编排时的分布式事务需要使用编排事务；注解十事务就类似于seata的那种方式，目前已停止更新，不推荐使用。

目前鸿鹄平台的分布式事务仅支持SAGA模式，通过设置红线节点来控制交易流程是否回滚（SAGA+红线节点）。

* 由业务方提供补偿服务（外调undo）或者方法（本地undo）
* 仅支持串行执行事务节点，出错或者未明不允许跳过，这样整个交易流程中只会出现一个失败或者未明的节点。
* 在红线节点及红线节点前，事务出错以后反向补偿前面节点。 **注意：如果依照这个理论，那么红线节点应该是后置处理节点前的那个节点**
* 在红线节点发生未明，经联机重试后仍未明，留待自检服务处理。自检服务重试红线节点，如果失败，则进行反向补偿；如果未明，发送事务差错模块；如果成功，可调用应用提供异步扩展接口帮助应用完成后面的后续功能，以确保最终一致性。
* 红线节点执行成功以后，后续节点如果失败，则不再向前补偿；后续节点未明，经联机重试后仍然未明的，留待自检服务处理。自检对于后续节点的失败和未明情况，可调用应用提供异步扩展接口完成后续功能，确保一致性。
* 幂等
* 锁
* 分库分表
* 规则引擎

运维态：

1. 分布式应用运维 T-CAM
2. 联机批量监控
3. 事务流水核对
4. 参数配置中心
5. 应用资源管理
6. 单元管理
7. 消息管理
8. 全局序号管理
9. 批量调度监控
10. 服务治理
11. 单元定位管理

运行态：

1. 单元定位 T-UPC

主要为企业级单元化架构下的 单元路由 提供 路由映射服务能力。

备注：路由映射就是 业务要素 到单元之间的映射关系。业务要素主要包括 账号、卡号、客户号。 本质就是根据分片键分片的过程。

1. 消息中心 T-MSC（TDMQ）
2. 统一序号发生器 T-SNG

提供可定制规则的发号服务，生成分布式场景下的全局唯一序号。

1. 分布式批量调度平台 T-DBSE

提供给分布式系统用于配置和部署作业任务的定时调度跑批

## 鸿鹄平台技术栈的使用原则：

开发态能用尽用；运维态必须使用；运行态按需使用；

## **中台相关术语介绍**

1. 单元化高可用

在单元化架构下为应用提供的高技术解决方案，主要包括：

* 支撑“四地八中心”
* 实现“同城双活异地可切换”
* 动态调拨流量
* 实现单元维度的灰度发布
* 实现应用监控（POMP）

1. 应用安全保障

* 服务的访问控制
* 敏感数据安全传输
* 敏感参数加密
* 敏感数据加解密
* 安全审计日志记录

1. 分布式批处理
2. 分布式事务一致性
3. 消息可靠传输
4. 微服务化支撑

* 通过注册中心进行服务的注册
* 微服务包括网关服务和应用服务
* 四地八中心企业级单元化架构下微服务之间的调用，以及限流、熔断、降级等能力

1. 一站式应用开发
2. 分布式应用运维治理
3. 数据访问代理

# 二，分布式开发技术栈介绍

## 1，消息中心的介绍

1.1 MQ的介绍

（1）**队列**是一种FIFO先进先出的数据结构。消息有生产者发送到MQ进行排队，然后按照原来顺序交由消费者进行处理。

（2）MQ的优点：异步、解耦、削峰填谷

（3）MQ的缺点：系统可用性降低、系统复杂性提高、消息一致性问题

**1.2消息中间价产品介绍**

1.2.1 RabbitMQ

**1.2.2 RocketMQ**

**1.2.3 TDMQ**

**1.2.4 kafka**

**1.3消息中心介绍**

**消息中心服务 在单元化架构下隶属于Gzone.**

1.3.1 功能介绍

**1.3.2 架构介绍**

**1.3.3 使用指引**

## **2，**序号发生器的介绍

序号发生器隶属于全局组件范围，不隶属于具体的某个单元。

2.1 术语说明

（1）雪花算法

（2）节点内递增序号

（3）全局连续递增序号

（4）全局递增序号

（5）指定规则序号（吉祥号）

（6）随即序号

（7）模板序号

（8）序号规则

**2.2 架构介绍**

**2.3 使用指引**

## **3，**单元定位组件介绍

单元定位组件隶属于 全局组件 的范围。

3.1 术语说明

**3.2 架构介绍**

**3.3 使用指引**

## **4，分布式事务的介绍**

### 4.1 分布式事务介绍

### 4.2 常见分布式事务的解决方案介绍

#### 4.2.1 XA规范方案

#### 4.2.2 AT模式方案

#### 4.2.3 柔性事物TCC方案

#### 4.2.4 可靠消息最终一致性方案

#### 4.2.5 最大努力通知方案

#### 4.2.6 SAGA方案

### 4.3 中台分布式事务组件介绍

#### 4.3.1 原理介绍

（1）当前分布式组件只支持SAGA模型的分布式事务

（2）当前分布式组件只能使用在企架规范的分布式系统内

（3）SAGA模型的分布式事务不具有隔离性，需要通过业务设计来解决。

（Seata时通过全局锁来解决隔离性问题的）

1. SAGA模型中一个全局事务是通过一系列子事务串连起来的；每个子事务必须提供补偿接口（undo）、状态查询接口（query）、补偿接口的状态查询接口（undoQuery）、异步扩展接口；
2. 正向执行接口，补偿接口，异步扩展接口都需要具备幂等性。

#### **4.3.2 整体架构介绍**

（1）分布式事务SDK

包括事务协调器和事务管理器。**事务协调器**负责对服务编排中配置事务的服务进行事务的发起，比如说在原子服务中提供本地事务协调机制，在组合服务中提供SAGA分布式事务协调机制。说人话就是负责事务的开启，事务的提交，事务的补偿的工作。**事务管理器**负责对事务流水信息进行记录。

1. 事务的自检服务：负责检查事务状态，进行事务补偿或者发送事务差错
2. 事务差错处理服务：包含差错数据存储、差错数据查询、差错数据告警
3. 事务流水库：需要在业务库中创建“全局事务流水表”和“分支事务流水表”
4. 事务差错库：中台创建

##### 4.3.2.1 事务管理流程

（1）正常的执行场景

（2）异常的场景

（3）超时场景

##### **4.3.2.2 事务流水核对**

准实时流水核对：通多定时轮询扫描事务流水表，找出业务状态不一致的事务，进行自动调整或者登记事务流水差错库交给人工处理。

1. 核对轮询窗口
2. 事务自检处理
3. 主流水处于“处理中”、子流水处于“成功”状态：内存上下文已销毁，自检服务的处理规则是调用原交易状态查询接口，如果原交易成功，则从当前节点反向补偿；如果原交易失败，则从上一个节点反向补偿；如果重试原交易超时或者原交易状态查询接口也超时，则将该事务流水写差错库，请求人工处理。
4. 主流水处于“补偿中”状态，子流水处于“补偿失败”状态
5. 主子流水均处于“补偿中”状态

#### **4.2.3 使用案例**

##### 4.2.3.1 开客户场景

##### **4.2.3.2 客户信息维护场景**

##### **4.2.3.3 客户合并场景**

##### **4.2.3.4 不规范客户一键恢复场景**

##### **4.2.3.5 BGL交易场景**

#### **4.2.4 使用指引**

##### 4.2.4.1 分布式事务配置

（1）配置事务类型

（2）配置补偿类型

（3）配置状态确认

（4）配置红线标记

（5）配置事务结束标记

（6）配置异步扩展功能

（7）分布式事务嵌套（跨组件调用）

（8）配置数据源

（9）配置数据库连接池

（10）创建流水表（主流水表+子流水表）

##### **4.2.4.2 补偿规则**

当前节点失败以后，从失败节点的上一个节点开始发起补偿。在补偿的过程中根据补偿方法返回的不同状态执行不同的逻辑。

1. 返回成功：继续补偿后续带补偿的节点
2. 返回业务类失败：终止补偿，进入差错库
3. 返回技术类失败：使用原交易的重试参数发起重试，重试成功，继续往后重试；重试失败，进入差错库
4. 返回未明：使用原接口正向执行方法的重试参数发起重试，重试成功，继续往后重试；重试失败，进入差错库

##### **4.2.4.3 自检服务**

通过定时轮询扫描事务流水记录，找出业务状态不一致的事务，进行重试或者人工处理

##### **4.2.4.4 差错处理**

（1）监控告警

（2）事务差错流水查询

（3）联机更新差错处理状态

（4）离线更新差错处理状态

##### **4.2.4.5 数据备份清理**

（1）清理：定时清理事务流水表和事务自检表

（2）备份：定时备份事务流水表

## 5，幂等的介绍

### 5.1 原理介绍

#### 5.1.1 术语介绍

幂等：请求一次和请求多次结果相同。

防重复提交：多次重复的提交到后台以后，系统能够对重复的提交进行去重。

总结：幂等是目的，放重复提交时手段

幂等组件：通过AOP切面来拦截请求，然后查询**幂等数据库**判断幂等状态。

幂等数据库：存储幂等流水表。

#### **5.1.2 幂等的处理流程**

系统每次收到请求以后，首先去幂等表中进行查询：

1. 如果未查到幂等流水记录，则向幂等表中插入一条流水记录，且初始状态为运行中；插入成功后，就执行真正的业务操作，返回结果时，更新幂等表的状态，返回结果信息；
2. 如果查询到的记录状态为非运行时，则直接返回记录中的结果信息，不执行业务流程。
3. 如果查询到的记录状态为运行时，则返回结果信息为“业务处理中”（这里其实就是状态未明）；此时由业务决定后续操作，是重试还是发起业务查询？

### **5.2 中台幂等组件介绍**

#### **5.2.1 幂等组件的功能**

#### **5.2.2 幂等使用指引**

## 6，分布式锁的介绍

## 7，交易路由的介绍

## 8，分库分表的介绍

## 9，微服务网关

## 10，联机框架

## **11，批量框架**

## **12，规则引擎**

## **13，数据源统一管理**

## **14，redis组件**

## **15，敏感参数的加密解密**

# **三，微服务治理技术栈**

## 1，网关分组和鉴权

## 2，服务限流

## 3，服务熔断

## 4，服务监控

## 5，接口防腐

# 四，服务运维

# 五，调度引擎