# **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

### 6.1 锁的介绍

### 6.2 进程锁的实现方案

### **6.3 中台分布式锁组件的介绍**

#### 6.3.1 原理介绍

锁持有人：当前ip+进程号+线程号+lockKey

lockKey: UUID

分布式锁表（distributed\_lock）包含字段：

锁名称、解锁钥匙、锁获取者信息，锁重入次数、锁超时时间、乐观锁版本，创建时间，更新下时间。

#### 6.3.2 整体架构介绍

##### 6.3.2.1 处理机制

（1）加锁原理

1. 加锁前需要将唯一资源id作为锁名称到数据库中去查询锁信息。
2. 如果没有查询到，存储加锁的数据（所信息+钥匙+锁持有者信息+超时时间）

如果存储成功，则锁加成功了；

如果存储失败，则锁加失败了；

1. 如果查询到了，则需要进一步判断锁拥有者与当前申请者是否一致？

如果不一致，此时校验锁超时的时间是否超时？

如果超时，则删除锁信息。

如果删除成功，则存储锁信息加锁；

如果删除失败，则加锁失败；

如果未超时，加锁失败了。

如果一致，则当前锁为重入锁，锁计数器+1，同时更新锁信息，更新时间修改。

如果更新成功，则锁加成功了；

如果更新失败，则锁加失败了；

1. 补充：如果锁加失败以后，是否进行重试？

如果重试，则重新开始新的加锁流程；

如果不重试，那就加锁失败吧。

（2）解锁原理

1. 使用钥匙+锁拥有者信息查询所信息
2. 如果没有查询到，那肯定解锁失败
3. 如果查询到锁信息，判断重入计数器是否小于等于0？

如果是，则删除锁记录；

如果删除成功，则解锁成功；

如果删除失败，则解锁失败；

如果不是，则当前锁为重入锁，重入计数器减1，解锁成功。

1. 补充：如果解锁失败，还可以进行重试。

##### **6.3.2.2 使用场景**

（1）正常场景

（2）超时场景

#### **6.3.3 使用指引**

##### （1）配置说明

选择数据源，可以配置TDSQL，则需要创建表distribute\_lock;

也可以配置redis。

##### （2）功能接口

## 7，交易路由的介绍

### 7.1 原理介绍

交易路由组件为网关和应用之间的流量调度和服务路由提供统一的解决方案。

交易转发：对跨单元交易提供转发的支持

就近路由：对同城数据中心之间的就近路由

流量调度：对数据中心按比例流量调度提供支持

容灾切换：对同城、异地两种跨数据中心容灾切换提供支持

## 8，分库分表的介绍

## 9，微服务网关

## 10，联机框架

### 10.1 原理介绍

#### 10.1.1 联机框架的介绍

联机框架是对微服务开发过程进行规范，统一开发和协作的工具。

联机框架提供了 脚手架 和 服务编排工具；另外也提供了权限管理和基线管理的功能，用于支持并行开发。

服务编排提供了可视化的流程编排工具，并整合和分布式事务，幂等等组件。

联机框架是微服务分布式开发框架的一部分，由开发平台设计器和服务编排引擎SDK组成。

#### **10.1.2 术语说明**

* 服务编排
* 脚手架
* 设计器
* 解析器
* 执行器
* 服务编排引擎
* 组件
* 服务
* 数据字典
* 接口
* TSF

### **10.2 整体架构介绍**

#### 10.2.1 逻辑架构

#### 10.2.2 功能架构

#### 10.2.3 处理机制

## **11，批量框架**

### 11.1 原理介绍

### 11.2 整体架构介绍

## **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**

## **16,分布式基础工具-报文统一处理组件**

### 16.1 原理介绍

该组件的目的是对新线系统的报文统一处理，比如说报文头的部分参数自动填充、跨组件访问身份鉴权值填充、企架报文头数据填充到上下文。

该组件的优点：

* 实现了应用系统、应用组件间联机交易报文的统一和标准化
* 实现了现状子系统和应用系统之间联机交易报文的格式转化。
* 实现了企业级的链路跟踪

该组件为应用侧的一个jar包，它隶属于基础组件范围，不隶属于某个具体的单元。

1. 当新线系统A发起联机交易时，使用该组件进行报文的统一处理与生成；
2. 新线系统B收到联机请求后，首先通过该组件对报文进行统一的处理与解析，然后在进入接口的业务逻辑中。
3. 业务逻辑处理结束后，使用该组件对响应报文进行统一处理和生成。
4. 新线系统A接收到响应后，首先通过该组件对B系统返回的响应进行统一处理与解析，然后继续A系统业务逻辑处理

### **16.2 整体架构**

#### 16.2.1 功能架构图

（1）报文的生成与解析

都生成了那些报文？5个流水号？  
 （2）跨组件访问身份认证鉴权

代码是如何鉴权的？

（3）将企架报文头中的相关数据项填充到交易上下文供技术中台的SDK使用

在什么位置填充到了上下文？

#### **16.2.2 功能流程图**

（1）组件间的调用过程

（2）组件内的调用过程

### **16.3 使用指引**

## **17，分布式基础工具-标准日志输出**

### 17.1 原理介绍

标准日志输出组件可以对新线系统的日志标准输出，为行内业务系统提供日志记录，分类和分级，开发人员和运维人员可以根据日志对对各应用平台进行跟踪与调试。

本组件隶属于基础组件范围，以jar包的形式提供给应侧（不需要独立部署）

### **17.2 整体架构**

#### 17.2.1 功能架构

该组件主要有以下功能：

1. 日志输出格式转换
2. 对日志中敏感信息进行屏蔽
3. 在网关打印流水日志（来自不同组件的请求才打印流水日志）

#### **17.2.2 功能流程**

（1）输出日志时首先判断格式，有就转换为指定格式，没有就原样输出。

（2）然后判断日志内容中是否存在敏感词

（3）日志信息获取：通过ServletFilter、base-msg、base管道，在回调中取日志中需要的信息放入MDC（这是个什么东西）。

（4）在网关服务中自动打印流水日志，也可以配置include和exclude的URL，对打印流水日志的URL进行控制；也可以通过判断报文reqHeader中的callCode(调用方编码)和targetSerCode(目标服务编码)前6位，对来自不同组件的请求才打印流水日志。

### **17.3 使用指引**

# **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