# 简洁升级概要设计



**锐明技术股份有限公司**

修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 版本号 | 描述 | 作者 |
| 2025年2月21日 | V2.16.5 |  | 陈刚 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 概述

## 编写目的

人工操作的效率极低，出错率也高，现在客户越来越多，现有的人力已经无法满足及时快速的无问题的给客户进行安装和升级。同时未来也希望将升级移交给其他团队（甚至可能是客户自己）执行，而不是由研发来对客户环境进行升级。

基于此对升级流程进行梳理简化，对升级流程进行自动化处理，满足简化升级；

## 专有名词

|  |  |  |
| --- | --- | --- |
| 序号 | 名词 | 解释 |
| 1 | 车组 | 在中台，车组为一抽象概念，可以是具体的车队、车组或组织等 |
| 2 | 司机 | 在中台，司机为一抽象概念，可以是具体的司机、从业人员、乘务员等 |
| 3 | 注册 | 设备上线必备流程，通过协议类型，设备编号等信息获取设备相关信息，包括设备authId(鉴权码) |
| 4 | 鉴权 | 808协议系列设备上线必备流程，在注册流程之后。通过设备authId校验设备合法，合法则返回设备相关信息 |
| 5 | base-mfec-sdk | MFEC为Model Field Expansion Capability的缩写，中文名称：模型字段扩展能力，此SDK主要提供基于中台现有模型的扩展能力封装，行业层引入此SDK后，可以方便的自定义扩展字段  此SDK为在firefly-V2.12.2内部改造去除原base-metadata-service组件时新增，改造为基于数据库本身json字段模型扩展能力方式 |
|  |  |  |
|  |  |  |

## 环境要求

|  |  |  |
| --- | --- | --- |
| 序号 | 限定类型 | 限定说明 |
| 1 | 安装环境 | Linux(CentOS7.9、Redhat8.3、Rocky linux 9.x、麒麟OS) |
| 2 | mariadb | 10.11.6 |
| 3 | Clickhouse | 24.8.4.13 |
| 4 | 必备中间件 | Redis/Nacos/Kafka/JDK(默认X86架构, 国产化环境需要选用ARM架构) |
| 5 | 可选中间件 | CK(含zookeeper)、flink |

## 软件特性

### 开发环境

1. linux平台：Centos 7.9 X64
2. 开发语言：JAVA、C++

### 限定条件

*{描述本方案的特殊限定条件}*

### 标准配置要求

*{描述本方案的标准服务器、硬件配置等要求}*

### 硬件配置要求

（1）单机模式：16C/64G

（2）集群模式：

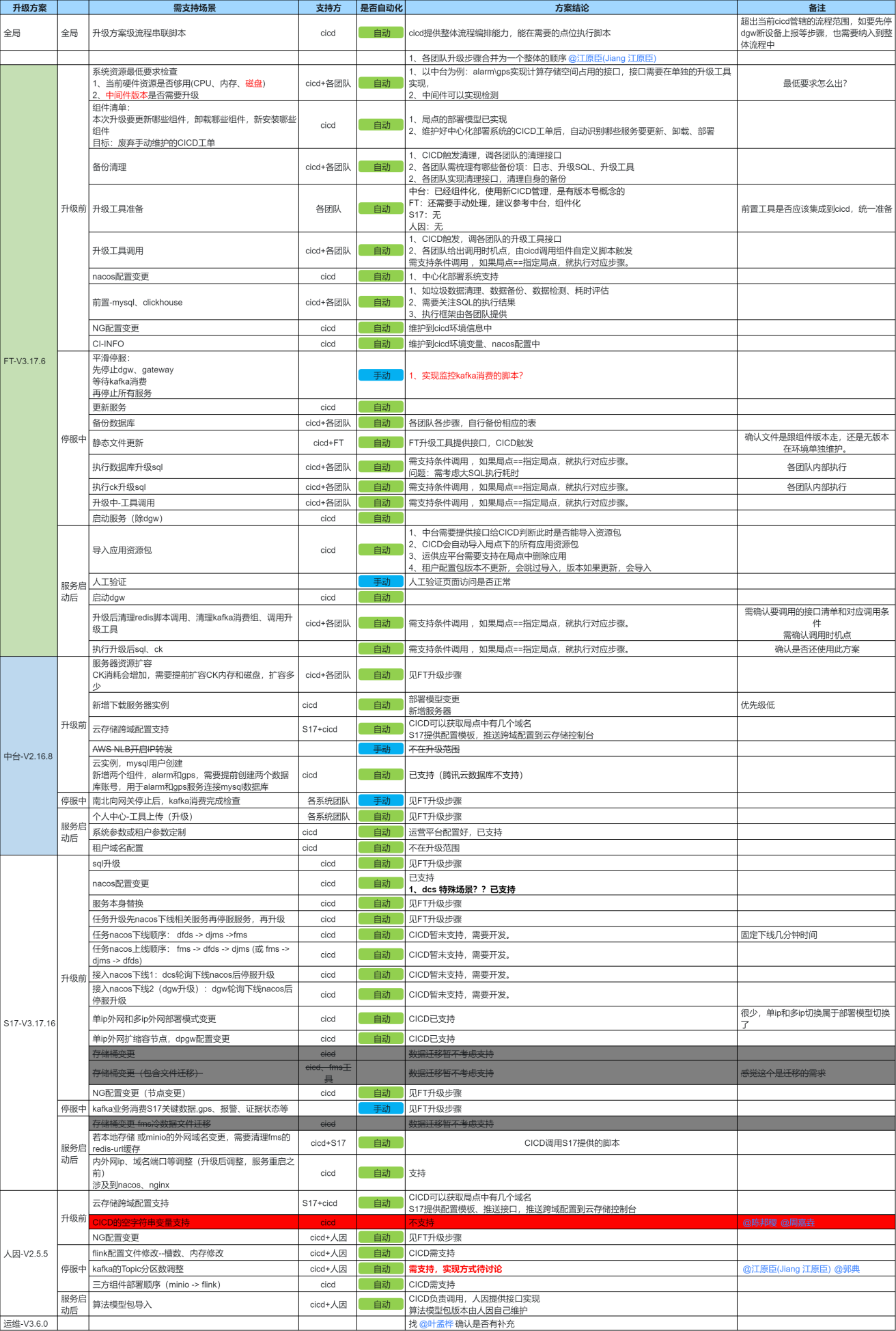
低于2000设备并发：4\*4C/16G（业务+数据库）

2000～10000设备并发：4\*8C/32G（业务）+3\*4C/16G（数据库）

10000～30000设备并发：4\*16C/64G（业务）+3\*8C/32G（数据库）

# 设计原则

# 场景分析



# 总体方案设计



<https://docs.qq.com/mind/DREZ0RVdPa0pvaU1P>

# 需求设计

## CICD

### 脚本切点

|  |  |
| --- | --- |
| 原因和动机 | 各组件升级时，需要在合适的时机点执行特定升级操作，如执行sql、调用升级工具接口等。由于目前cicd不具有整体流程编排开放给各团队自行编写的能力，因此cicd将在合适时机点，调用各组件的自定义脚本。这些调用时机点就是自定义脚本切点。 |
| 功能描述 | 时机点清单：   1. 升级组件安装/升级   在所有业务组件安装前，需要执行前置检查，而前置检查需要在升级组件上运行。因此需要在前置检查前安装各团队的升级组件。   1. 前置系统资源检查   在处理升级动作前，需要确保系统资源满足要求，资源检查作为第一个前置检查进行。所有组件的资源要求均上报，综合满足后，才会进入下一步，否则终止升级。   1. 前置业务检查   完成系统资源检查后，各组件会进行业务检查，包括业务数据合法性检查等。等所有组件的业务检查均通过，才会进入下一步，否则终止升级。   1. 前置升级处理   确认环境符合升级条件后，各组件将进行前置升级处理。对于不需要停服即可执行的操作，以及停服前的准备操作，均可放在此处完成。处理完此步，即可进行优雅停服，然后进入组件新版本安装步骤。   1. 启动前处理   在组件新版本均替换完成后，不会立刻启动服务，而是进行启动前处理。此步是最主要的升级处理，如不兼容sql执行等均在此处理完成。此步处理完，即可进入全系统的优雅启动步骤。   1. 启动后处理   全环境各组件启动后，即可执行有组件依赖的升级处理，如导入资源包等。此步处理完后，即可进行核心北向功能人工验证。   1. 南向开放操作   人工验证完成后，即可开放南向入口，使全系统开始加载全部工作负载。开放南向后，系统视作停服升级完成，可正式恢复正常，可以交由客户使用。   1. 升级后处理   当停服升级完成后，如有需持续处理的数据，可在此步骤完成。 |
| 优先级 |  |
| 前置条件 |  |
| 功能输入 |  |
| 功能输出 |  |
| 正常流程 | 参见总体方案流程 |
| 异常流程 |  |
| 原型图 |  |
| 性能指标 |  |
| 约束条件 |  |
| 补充说明 |  |

### 自定义脚本基础设施

|  |  |
| --- | --- |
| 原因和动机 | 为了执行各团队各组件的升级方案，cicd需要提供组件自定义脚本能力。自定义脚本除能执行代码外，还需要使用一些公共功能、公共接口，这些接口由cicd提供。 |
| 功能描述 | 1. 脚本语言   cicd提供python作为脚本语言。组件可在项目中打包要执行的python脚本，如放在config等目录，并在组件描述符中，配置某个切点执行某个脚本的某个方法。   1. 依赖引用方式   使用自定义python脚本时，可引用各类基础设施模块，cicd提供三种可引用模块：  cicd公共功能模块  预定义三方模块  脚本多文件互引   1. 环境、升级信息   cicd在cicd公共功能模块提供环境的信息，如局点/环境id，主机-组件关系，主机信息，多域名清单，升级的起始版本目标版本等。   1. http客户端   cicd提供python可调用的http客户端库，以便脚本调用各团队升级服务的接口。   1. sql能力   待讨论确认，各团队使用自身升级服务完成sql执行，还是需要在脚本中执行sql。 |
| 优先级 |  |
| 前置条件 |  |
| 功能输入 |  |
| 功能输出 |  |
| 正常流程 |  |
| 异常流程 |  |
| 原型图 |  |
| 性能指标 |  |
| 约束条件 |  |
| 补充说明 |  |

### 系统资源检测

|  |  |
| --- | --- |
| 原因和动机 | 升级过程可能占用额外的系统资源，包括内存、磁盘等。如备份会占用磁盘，升级服务启动会占用内存，执行会占用cpu。为避免系统资源不足，导致升级无法完成，需要提前检测系统空闲资源是否满足要求。 |
| 功能描述 | cicd将收集各组件的系统资源要求，汇总后与当前系统资源做比对，如果不满足要求，则终止升级。  cicd定义python脚本的接口，来收集各组件的资源需求。有需要的组件，应实现脚本接口，并在组件描述符申明需要上报资源消耗。  需检测的系统资源：  内存、磁盘空间 |
| 优先级 |  |
| 前置条件 |  |
| 功能输入 |  |
| 功能输出 |  |
| 正常流程 |  |
| 异常流程 |  |
| 原型图 |  |
| 性能指标 |  |
| 约束条件 |  |
| 补充说明 |  |

### flink配置修改

|  |  |
| --- | --- |
| 原因和动机 | 人因要求在升级时修改flink配置，cicd目前不支持修改中间件配置。 |
| 功能描述 | 需确认使用场景，是大部分升级都要改，还是就这一次  需确认要改那些配置，设计 |
| 优先级 |  |
| 前置条件 |  |
| 功能输入 |  |
| 功能输出 |  |
| 正常流程 |  |
| 异常流程 |  |
| 原型图 |  |
| 性能指标 |  |
| 约束条件 |  |
| 补充说明 |  |

### nacos配置修改

|  |  |
| --- | --- |
| 原因和动机 | 目前cicd只在私有化支持nacos配置管理，需要考虑方案在中心化方式下维护nacos配置和变更。 |
| 功能描述 | 考虑升级只替换application.properties?  update放到日常维护中改？ |
| 优先级 | 低 |
| 前置条件 |  |
| 功能输入 |  |
| 功能输出 |  |
| 正常流程 |  |
| 异常流程 |  |
| 原型图 |  |
| 性能指标 |  |
| 约束条件 |  |
| 补充说明 |  |

### ng配置修改

|  |  |
| --- | --- |
| 原因和动机 | 目前cicd在升级时，不会推送ng配置，如果ng配置修改，需要单独推送。需要考虑在升级时推送ng配置。 |
| 功能描述 |  |
| 优先级 |  |
| 前置条件 |  |
| 功能输入 |  |
| 功能输出 |  |
| 正常流程 |  |
| 异常流程 |  |
| 原型图 |  |
| 性能指标 |  |
| 约束条件 |  |
| 补充说明 |  |

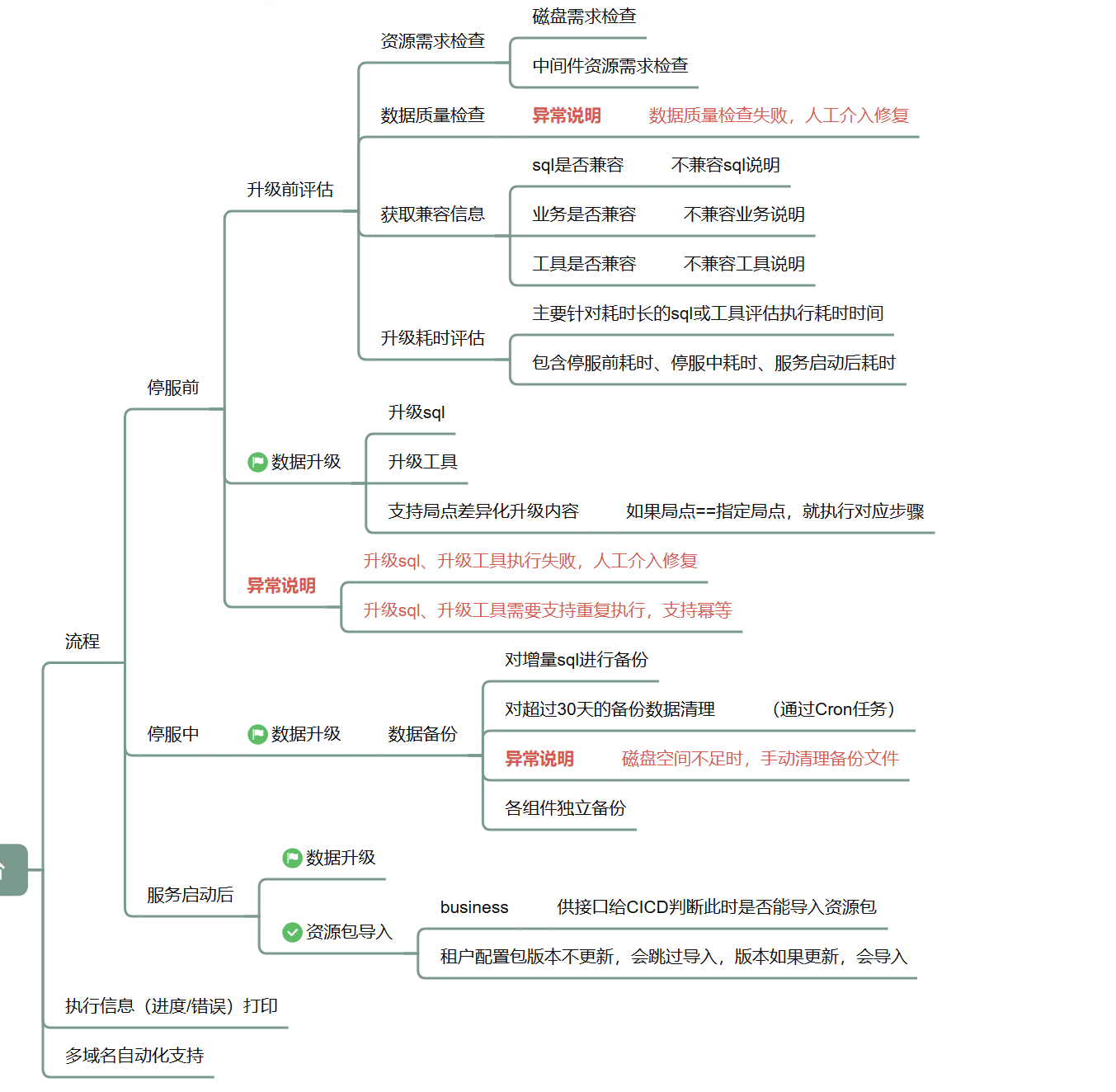
### 打包方案

|  |  |
| --- | --- |
| 原因和动机 | 此次调整，导致cicd需要携带额外的配置文件和信息，因此打包方案需要调整。 |
| 功能描述 |  |
| 优先级 |  |
| 前置条件 |  |
| 功能输入 |  |
| 功能输出 |  |
| 正常流程 |  |
| 异常流程 |  |
| 原型图 |  |
| 性能指标 |  |
| 约束条件 |  |
| 补充说明 |  |

### 日志方案

|  |  |
| --- | --- |
| 原因和动机 | 本升级方案交给客户执行后， 如果出现问题，如何让客户快速定位问题，或拿回日志，方便研发排查问题。 |
| 功能描述 |  |
| 优先级 |  |
| 前置条件 |  |
| 功能输入 |  |
| 功能输出 |  |
| 正常流程 |  |
| 异常流程 |  |
| 原型图 |  |
| 性能指标 |  |
| 约束条件 |  |
| 补充说明 |  |

## 中台



### 资源需求检查

|  |  |
| --- | --- |
| 原因和动机 | 在执行升级操作前，需对服务器资源（磁盘、CPU、内存）进行评估，确保升级工具、SQL脚本执行以及增量备份等操作有足够的资源支持，避免因资源不足导致升级失败或系统异常。 |
| 功能描述 | 在升级前检查以下资源需求：   1. 升级工具运行资源：CPU、内存。 2. 增量备份SQL资源：磁盘空间（需预估备份文件大小）。 3. 数据双写资源（如适用）：额外的磁盘空间和I/O负载。 |
| 优先级 | 低 |
| 前置条件 | 1、升级工具已部署或准备就绪。  2、服务器资源监控工具可用（如 df、free、top）。 |
| 功能输入 | 当前版本、目标版本、envInfo环境信息 |
| 功能输出 | 1. 升级工具服务器资源评估    1. 服务器当前空闲 CPU 和内存    2. 预估升级工具的内存占用（通过jvm配置的xmx）    3. 预估升级工具的CPU占用（累加每个升级脚本的资源）    4. 评估结果：通过 2. 增量备份SQL资源    1. 备份数据库表需要的磁盘空间    2. 评估结果：通过 3. 数据双写资源评估    1. 空闲磁盘空间    2. 原始数据空间大小、双写需要的空间大小    3. 评估结果：通过 4. 评估结果说明    1. 通过：资源充足，可继续升级。    2. 警告：资源紧张，需人工确认是否继续。    3. 失败：资源不足，终止升级流程。 |
| 正常流程 | 正常流程：   1. 升级工具需要的cpu、内存评估；    1. 升级工具进程的CPU占用（如JVM配置的线程数）。    2. 内存需满足升级工具的最小要求（如 -Xms512m -Xmx2g）。    3. 升级工具所需临时空间（如解压包、日志文件） 2. 增量备份sql需要磁盘评估；    1. 评估MySQL和ClickHouse在执行增量备份时所需的磁盘空间、CPU和内存资源，确保备份操作不会因资源不足而失败。    2. 占用 ≈ 当前数据总大小（data\_length + index\_length）\* 1.5 3. 数据双写需要磁盘空间评估；    1. 远程评估ClickHouse数据双写磁盘空间方案场景：工具服务与ClickHouse服务器分离，需远程查询CK表空间占用，评估双写磁盘需求。    2. 双写磁盘空间 = 原始数据大小 × 2（双副本） + 临时缓冲区（10%~20%）。 4. 生成报告：输出资源检查结果。 |
| 异常流程 | 1. 磁盘不足：    1. 提示管理员手动扩展磁盘或终止升级，清理临时文件（如日志、缓存）。 2. CPU/内存不足：    1. 建议调整升级工具配置（如减少并发线程）。    2. 终止升级并报警。 |
| 原型图 |  |
| 性能指标 | 1. 资源检查耗时：≤ 10秒。 |
| 约束条件 |  |
| 补充说明 | 1. 磁盘空间公式：    1. 总需求 = 升级工具空间 + 增量备份空间 2. 数据双写场景的特殊处理    1. 若升级涉及数据双写（如新旧版本并行写入），需额外检查：    2. 磁盘写入性能（IOPS）。    3. 双写期间的磁盘空间需求：    4. 双写磁盘需求 = 单日数据增量 × 双写天数 × 2示例阈值： 3. 示例阈值    1. CPU空闲 ≥ 20%。    2. 内存剩余 ≥ 1GB。    3. 磁盘剩余 ≥ 备份文件大小的2倍。 |

### 数据质量检查

|  |  |
| --- | --- |
| 原因和动机 | 在版本升级前，需确保数据质量符合目标版本的要求，避免因脏数据、格式不兼容或业务规则变更导致升级失败或数据异常。 |
| 功能描述 | 1. 版本适用性检查：    1. 支持配置版本范围（如 2.16.5~2.16.9），仅当升级源版本或目标版本在范围内时执行检查。 2. 数据质量检查项：    1. 脏数据检测（如空值、非法字符、违反唯一约束）。    2. 数据兼容性（如字段类型、枚举值变更）。    3. 业务规则校验（如金额不能为负、时间戳范围）。 |
| 优先级 | 中（影响升级成功率，但非所有版本均需检查）。 |
| 前置条件 | 1. 数据库连接信息已配置。 2. 目标版本的数据质量规则已定义（如通过配置文件）。 |
| 功能输入 | 升级源版本、升级目标版本、环境信息 |
| 功能输出 | 1. 、通过：所有检查项均符合阈值。 2. 警告：部分检查项超出阈值，但可继续升级（如记录日志）。 3. 失败：关键检查项不通过（如主键冲突），终止升级。 4. **报告示例：**   [数据质量检查报告]  1. 检查项: 必填字段空值检测  - 结果: 失败（发现 15 条空值记录，阈值 0）  - 建议: 执行 `UPDATE orders SET user\_id=0 WHERE user\_id IS NULL;`  2. 检查项: 金额负数检测  - 结果: 通过（0 条异常记录） |
| 正常流程 | 1. 解析版本范围：    1. 若当前版本或目标版本在 min\_version~max\_version 范围内，执行检查。 2. 执行SQL检查：    1. 运行配置的SQL语句，统计异常记录数。 3. 比对阈值：    1. 若异常数 ≤ threshold，标记为通过。    2. 生成报告：输出通过/警告/失败结果。   descript |
| 异常流程 | 1. 版本不匹配：跳过检查并记录日志。 2. SQL执行失败：终止检查并报错（如语法错误或表不存在）。 3. 数据库连接超时：重试3次后终止升级。 4. 数据库不存在，跳过检查并记录日志 |
| 原型图 |  |
| 性能指标 |  |
| 约束条件 | 1. 、仅支持SQL兼容的数据库（MySQL、ClickHouse等）。 2. 需提前定义检查规则，不支持动态生成SQL。 |
| 补充说明 | 1、脏数据手动执行修复脚本； |

### 获取兼容信息

|  |  |
| --- | --- |
| 原因和动机 | 在数据升级过程中，需明确每条SQL脚本和升级工具的兼容性范围，避免因版本不匹配导致升级失败或数据损坏。通过预检兼容性，可决定是否支持不停服滚动升级或必须停服升级。 |
| 功能描述 | 1. 标记兼容版本：    1. 为每条SQL脚本和升级工具标注兼容的最小版本（如 compatible\_version: 2.16.7）。 2. 兼容性判断：    1. 根据源版本和目标版本，判断是否需要执行该SQL或工具。    2. 规则：仅当 源版本 >= compatible\_version 时兼容。 3. 不兼容处理：    1. 输出不兼容的SQL/工具列表及说明。 4. 升级模式决策：    1. 兼容版本：支持不停服滚动升级。    2. 不兼容版本：必须停服升级。 |
| 优先级 | 高（直接影响升级策略和成功率） |
| 前置条件 | 1. 所有SQL脚本和工具已标注兼容版本（如通过元数据文件）。 2. 源版本和目标版本已明确。 |
| 功能输入 | 1. 源版本号（如 2.16.6）。 2. 目标版本号（如 2.16.9）。 |
| 功能输出 | 兼容报告  [兼容性检查结果]  1. SQL: update\_V2.16.7.sql  - 兼容性: 通过（源版本 2.16.6 >= 兼容版本 2.16.7? ❌ 不兼容）  - 影响: 修改orders表结构  - 建议: 需停服升级  2. 工具: migrate\_data\_tool.sh  - 兼容性: 通过（源版本 2.16.6 >= 兼容版本 2.16.5? ✅ 兼容）  - 影响: 迁移用户历史数据  升级模式：不停服滚动升级 或 停服升级 |
| 正常流程 | 1. 解析版本号：    1. 提取源版本（src\_version）和目标版本（dst\_version）。 2. 遍历检查SQL/工具：    1. 对每条记录，检查 src\_version >= compatible\_version。 3. 生成报告：    1. 输出兼容性结果和升级模式建议。 |
| 异常流程 | 1. 版本号格式错误：    1. 终止并提示“版本号格式不合法”。 2. 未标注兼容版本：    1. 默认视为不兼容，要求人工确认。 |
| 原型图 |  |
| 性能指标 | 1. 检查耗时 ≤ 1秒（百级脚本规模）。 2. 支持并发检查。 |
| 约束条件 | 1. 版本号需符合语义化版本规范（如 2.16.7）。 2. 仅支持预定义的兼容性规则，不支持动态推断。 |
| 补充说明 |  |

### 升级耗时评估

|  |  |
| --- | --- |
| 原因和动机 | 在升级过程中，准确评估关键SQL和工具的执行耗时，有助于：   1. 制定合理升级窗口：避免因执行超时影响业务连续性。 2. 优化资源分配：识别长耗时任务，优先并行化处理。 3. 风险预警：提前发现潜在性能瓶颈，减少升级失败风险。 |
| 功能描述 | 1. 评估范围：    1. 仅评估耗时较久的SQL和工具（执行时间 > 10秒）。    2. 分阶段统计：停服前、停服中、服务启动后。 2. 评估方法：    1. 基于数据量（GB/MB/条）和配置处理速率（如 50MB/s）动态计算耗时。    2. 支持执行中动态校准（实际速率偏差 >20%时调整预估）。 3. 输出建议：    1. 分阶段耗时报告 + 长耗时任务优化建议（如拆分或并行化）。 |
| 优先级 | 中（影响升级窗口规划，但非阻塞性） |
| 前置条件 | 1. 升级任务已标记阶段。 2. 数据量可获取（通过查询或配置）。 |
| 功能输入 | 源版本、目标版本、数据源、环境信息  SQL/工具的阶段、数据量、速率配置 |
| 功能输出 | [升级耗时评估]  1. 停服前阶段（并行）：  - migrate\_user\_data.sql: 100GB @ 50MB/s ×4 → 8分32秒  2. 停服中阶段（串行）：  - alter\_orders.sql: 3分20秒  3. 服务启动后阶段：  - rebuild\_indexes.sh: 4分10秒  ----------------------------------  总耗时: 15分02秒 | 关键任务: migrate\_user\_data.sql  [动态调整] 实际速率较预估+12%，已修正后续任务耗时 |
| 正常流程 | 1. 初始化检查    1. 校验输入参数完整性（版本号、数据源连接、任务元数据）。    2. 确认数据量可获取（通过查询或配置）。 2. 分阶段耗时计算    1. 停服前阶段：并行任务取最大耗时。    2. 停服中阶段：串行任务累加耗时。    3. 服务启动后阶段：非阻塞任务单独统计 3. 动态校准    1. 每30秒采集实际处理量，计算实时速率    2. 若 |actual\_speed - configured\_speed| / configured\_speed > 0.2，则修正剩余任务预估： 4. 报告生成    1. 按阶段输出耗时，标记超阈值任务（如 >5分钟）。    2. 提供优化建议（如增加并行度、拆分大表）。 |
| 异常流程 |  |
| 原型图 |  |
| 性能指标 |  |
| 约束条件 |  |
| 补充说明 |  |

### 数据升级工具

|  |  |
| --- | --- |
| 原因和动机 | 在CICD流程中，各组件需提供标准化的数据升级工具，由CICD在升级的各个阶段调用，确保：   1. 流程自动化：减少人工干预，提升升级效率。 2. 执行可控性：支持分阶段（停服前/中/后）执行和编排。 3. 幂等与可重入：处理执行中断或重复升级场景。 |
| 功能描述 | 1. 数据升级操作：    1. 数据备份：全量/增量备份源数据。    2. SQL执行：执行版本相关的DDL/DML脚本。    3. 工具调用：数据清理、数据转换、校验等。 2. 阶段划分：    1. 停服前：非阻塞操作（如备份、预检查）。    2. 停服中：阻塞性操作（如表结构变更、数据迁移）。    3. 服务启动后：后置任务（如索引重建、数据一致性校验）。 3. 核心特性：    1. 编排能力：通过配置定义执行顺序和依赖。    2. 幂等执行：记录执行状态，避免重复执行。    3. 断点续升：异常中断后可从断点继续。 |
| 优先级 | 高（直接影响升级流程的可靠性和自动化程度） |
| 前置条件 | 1. 数据源连接信息（IP/端口/认证）已配 2. 升级脚本和工具已按版本规范存放（如V2.16.7/upgrade/） |
| 功能输入 | fromVersion：升级起始版本（示例：V2.16.6）  toVersion：升级目标版本（示例：V2.16.7或V2.16.7-P1-T1）  src/dst：源/目标数据源（可相同）（示例：MySQL@10.0.0.1:3306）  stage：执行阶段（pre\_stop/in\_stop/post\_start）（示例：in\_stop）  envInfo：局点环境信息（如集群ID）（示例：{"cluster\_id":"bj-01"}） |
| 功能输出 | 执行摘要日志：  [SUCCESS] 执行完成：alter\_orders\_table.sql（耗时：2分30秒）  [SKIPPED] 已跳过：backup\_legacy\_data.sh（历史记录显示已成功）  状态记录：  数据库记录（如upgrade\_log表），字段包括：task\_id, version\_from, version\_to, stage, task\_name, status, start\_time, end\_time |
| 正常流程 | 1. 初始化：    1. 校验输入参数合法性（版本号、数据源连通性）    2. 查询upgrade\_log表，过滤已成功任务 2. 任务编排：    1. 根据配置按顺序执行当前阶段任务 3. 执行与监控：    1. 每个任务执行后更新状态（成功/失败）    2. 失败时终止流程并记录断点 4. 清理与报告：    1. 释放临时资源（如关闭数据库连接）    2. 输出执行摘要日志 |
| 异常流程 | 1. 任务执行失败：    1. 回滚当前任务（如SQL事务）    2. 记录失败状态    3. 终止流程并告警 2. 重复执行检测：    1. 跳过upgrade\_log中状态为SUCCESS的任务： |
| 原型图 |  |
| 性能指标 | 任务调度延迟：≤50ms/任务  状态查询响应：≤10ms  最大并行任务数：≤CPU核心数的80% |
| 约束条件 | 1. 版本规范：版本号需符合语义化（如V2.16.7-P1） 2. 数据库支持：需支持事务（如MySQL、PostgreSQL） 3. 工具兼容性：仅支持Linux/Windows可执行脚本或Python/Java工具 |
| 补充说明 | 1. 日志打印参考日志方案 2. 输出执行摘要日志，升级人员通过日志查看升级进度； 3. 输出详细日志，方便问题排查； 4. 功能增强说明在原有数据升级工具基础上，增加对局点差异化升级内容的支持，通过局点ID过滤需要执行的SQL和工具。 |

## FT

### 资源需求检查

|  |  |
| --- | --- |
| 原因和动机 |  |
| 功能描述 |  |
| 优先级 |  |
| 前置条件 |  |
| 功能输入 |  |
| 功能输出 |  |
| 正常流程 |  |
| 异常流程 |  |
| 原型图 |  |
| 性能指标 |  |
| 约束条件 |  |
| 补充说明 |  |

### 数据质量检查

|  |  |
| --- | --- |
| 原因和动机 | 在版本升级前，需确保数据质量符合目标版本的要求，避免因脏数据、格式不兼容或业务规则变更导致升级失败或数据异常。 |
| 功能描述 | 1. 版本适用性检查：    1. 支持配置版本范围（如 2.16.5~2.16.9），仅当升级源版本或目标版本在范围内时执行检查。 2. 数据质量检查项：    1. 脏数据检测（如空值、非法字符、违反唯一约束）。    2. 数据兼容性（如字段类型、枚举值变更）。    3. 业务规则校验（如金额不能为负、时间戳范围）。 |
| 优先级 | 中（影响升级成功率，但非所有版本均需检查）。 |
| 前置条件 | 1. 数据库连接信息已配置。 2. 目标版本的数据质量规则已定义（如通过配置文件）。 |
| 功能输入 | 升级源版本、升级目标版本、环境信息 |
| 功能输出 | 1. 、通过：所有检查项均符合阈值。 2. 警告：部分检查项超出阈值，但可继续升级（如记录日志）。 3. 失败：关键检查项不通过（如主键冲突），终止升级。 4. **报告示例：**   [数据质量检查报告]  1. 检查项: 必填字段空值检测  - 结果: 失败（发现 15 条空值记录，阈值 0）  - 建议: 执行 `UPDATE orders SET user\_id=0 WHERE user\_id IS NULL;`  2. 检查项: 金额负数检测  - 结果: 通过（0 条异常记录） |
| 正常流程 | 1. 解析版本范围：    1. 若当前版本或目标版本在 min\_version~max\_version 范围内，执行检查。 2. 执行SQL检查：    1. 运行配置的SQL语句，统计异常记录数。 3. 比对阈值：    1. 若异常数 ≤ threshold，标记为通过。    2. 生成报告：输出通过/警告/失败结果。   descript |
| 异常流程 | 1. 版本不匹配：跳过检查并记录日志。 2. SQL执行失败：终止检查并报错（如语法错误或表不存在）。 3. 数据库连接超时：重试3次后终止升级。 4. 数据库不存在，跳过检查并记录日志 |
| 原型图 |  |
| 性能指标 |  |
| 约束条件 | 1. 、仅支持SQL兼容的数据库（MySQL、ClickHouse等）。 2. 需提前定义检查规则，不支持动态生成SQL。 |
| 补充说明 | 1、脏数据手动执行修复脚本； |

### 获取兼容信息

|  |  |
| --- | --- |
| 原因和动机 |  |
| 功能描述 |  |
| 优先级 |  |
| 前置条件 |  |
| 功能输入 |  |
| 功能输出 |  |
| 正常流程 | 正常流程： |
| 异常流程 |  |
| 原型图 |  |
| 性能指标 |  |
| 约束条件 |  |
| 补充说明 |  |

### 升级耗时评估

|  |  |
| --- | --- |
| 原因和动机 |  |
| 功能描述 |  |
| 优先级 |  |
| 前置条件 |  |
| 功能输入 |  |
| 功能输出 |  |
| 正常流程 | 正常流程： |
| 异常流程 |  |
| 原型图 |  |
| 性能指标 |  |
| 约束条件 |  |
| 补充说明 |  |

### 数据升级

|  |  |
| --- | --- |
| 原因和动机 |  |
| 功能描述 |  |
| 优先级 |  |
| 前置条件 |  |
| 功能输入 |  |
| 功能输出 |  |
| 正常流程 | 正常流程： |
| 异常流程 |  |
| 原型图 |  |
| 性能指标 |  |
| 约束条件 |  |
| 补充说明 |  |

### 个人中心-工具上传自动化

|  |  |
| --- | --- |
| 原因和动机 |  |
| 功能描述 |  |
| 优先级 |  |
| 前置条件 |  |
| 功能输入 |  |
| 功能输出 |  |
| 正常流程 | 正常流程： |
| 异常流程 |  |
| 原型图 |  |
| 性能指标 |  |
| 约束条件 |  |
| 补充说明 |  |

## crocus

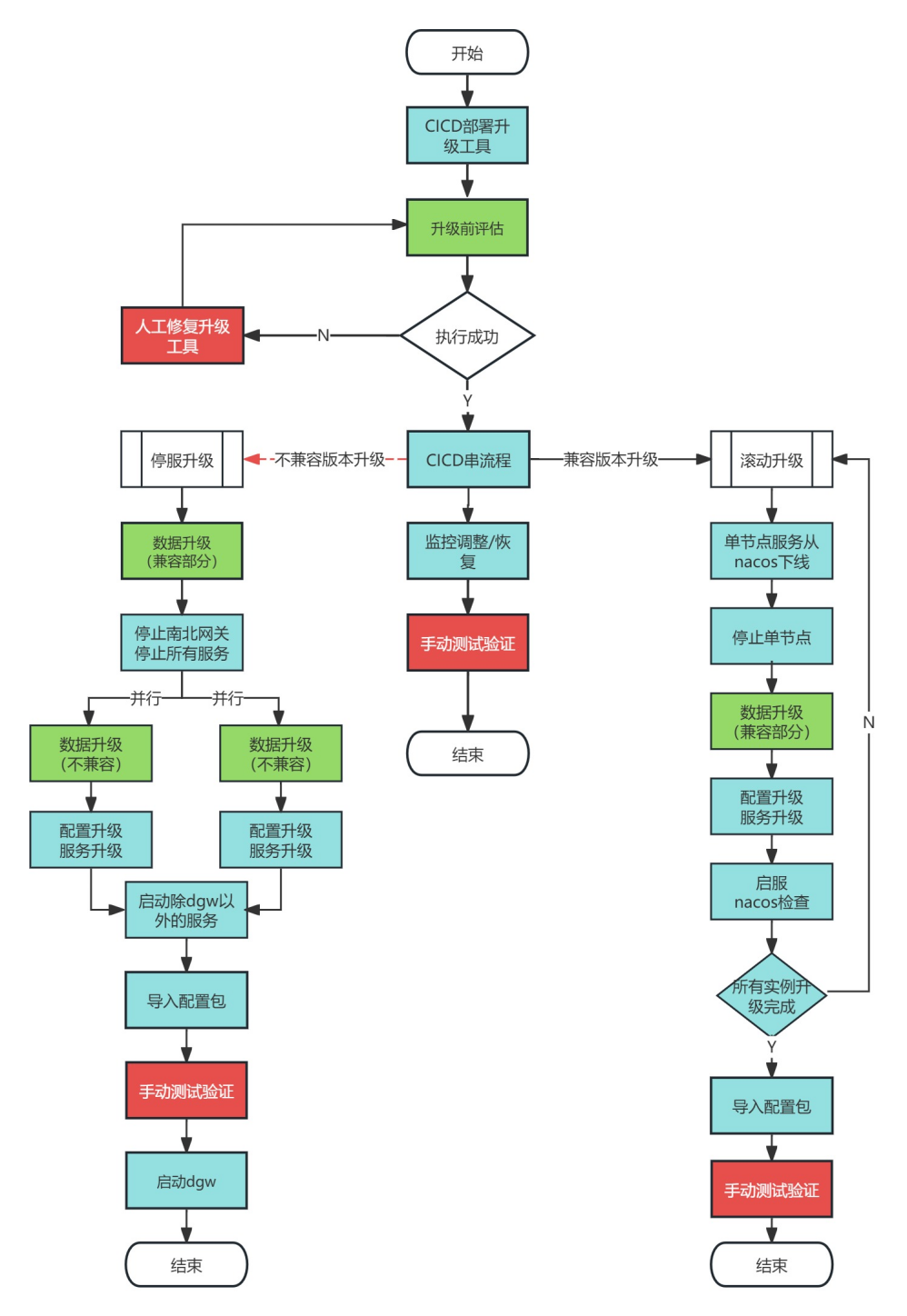
## s17

## 人因

# 中台（待定）

## 整体方案

### [升级流程](https://www.processon.com/v/6881d4d71284fd38ea967965?cid=67b54e00d8d98d4f63324eac)

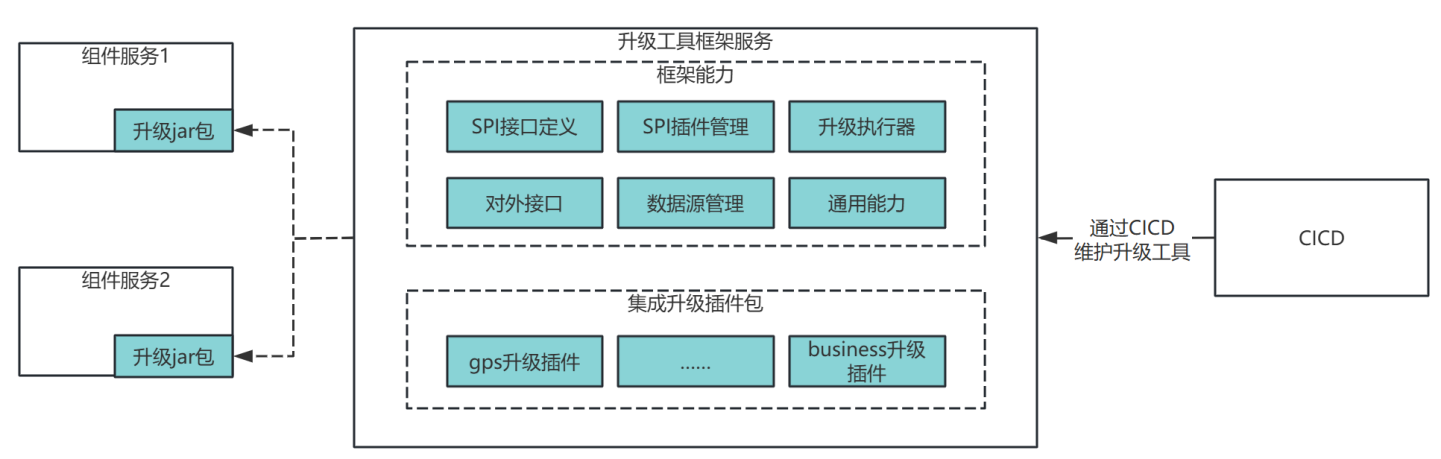


绿色部分是CICD串流程切点，切点处调用升级工具提供的接口能力；

蓝色部分是CICD串流向需要执行的内容；

红色部分人工介入修复工具和测试验证；

### [升级工具](https://www.processon.com/v/6881d5252bf6554159f43974?cid=685943c0c7884e2e72b73cfb)



升级工具提供统一的升级能力，由各组件实现升级工具，升级工具框架集成各组件提供的升级插件；

升级工具通过CICD管理维护；

#### 功能说明

1. **升级插件管理**
   1. 包含SPI接口定义和SPI插件管理，支持插件的动态加载和卸载，确保系统的灵活性和可扩展性。
2. **数据源管理**
   1. 管理源数据源和目标数据源的列表，支持每个组件单独设置数据源。
   2. 各插件根据数据源类型匹配相应的连接配置。
   3. 示例配置：

datasource:

srcServerMysql:

type: "server"

dstServerClickhouse:

type: "clickhouse"

1. **升级执行器**
   1. 负责执行从当前版本到目标版本的SQL和工具，确保在升级过程中所有操作的顺序和依赖关系得到满足。
2. **通用升级工具**
   1. 提供待定的通用功能，支持不同场景下的升级需求，如数据迁移、格式转换等。
3. **对外提供接口能力**
   1. 提供升级评估、数据升级等操作的API接口，支持外部系统的集成和调用。

#### 插件能力

升级插件是整个升级框架的核心，提供以下功能模块：

1. **资源需求检查**
   1. 磁盘需求检查：评估升级过程中所需的磁盘空间，包括备份、临时文件和增量数据的存储需求。
   2. 中间件资源需求检查：检查数据库、缓存、消息队列等中间件的资源使用情况，确保在升级过程中不会因资源不足导致服务中断。
2. **数据质量检查**
   1. 脏数据检测：识别数据中的空值、格式错误等问题，确保数据的完整性和有效性。
   2. 重复数据检测：检查唯一性约束，防止因重复数据导致的业务逻辑错误。
   3. 逻辑错误检测：验证字段值范围和业务规则，确保数据符合预期的逻辑关系。
3. **版本兼容性检查**
   1. SQL兼容性：检查待执行的SQL脚本与当前版本的兼容性，确保在升级过程中不会引入不兼容的变更。
   2. 需求兼容性：评估新版本对现有功能的影响，确保升级后系统的稳定性。
4. **升级耗时评估**
   1. 针对耗时较长的SQL或工具，评估其执行时间，帮助制定合理的升级窗口。
   2. 通过数据量和配置的处理速率动态计算预估耗时，并支持执行过程中的动态调整。
5. **数据升级工具**
   1. 提供增量SQL备份、SQL脚本、数据清理和数据刷写等功能，确保数据在升级过程中的安全和一致性。

#### CICD维护

升级工具通过CICD进行版本维护，确保每次升级都能在稳定的环境中进行。版本控制：所有升级脚本和工具通过版本控制系统管理，确保可追溯性和一致性。

自动化部署：通过CICD管道自动化执行升级流程，减少人工干预，提高效率和准确性。

监控与反馈：在升级过程中实时监控各项指标，及时反馈升级状态，确保问题能够快速定位和解决。

#### 开发流程

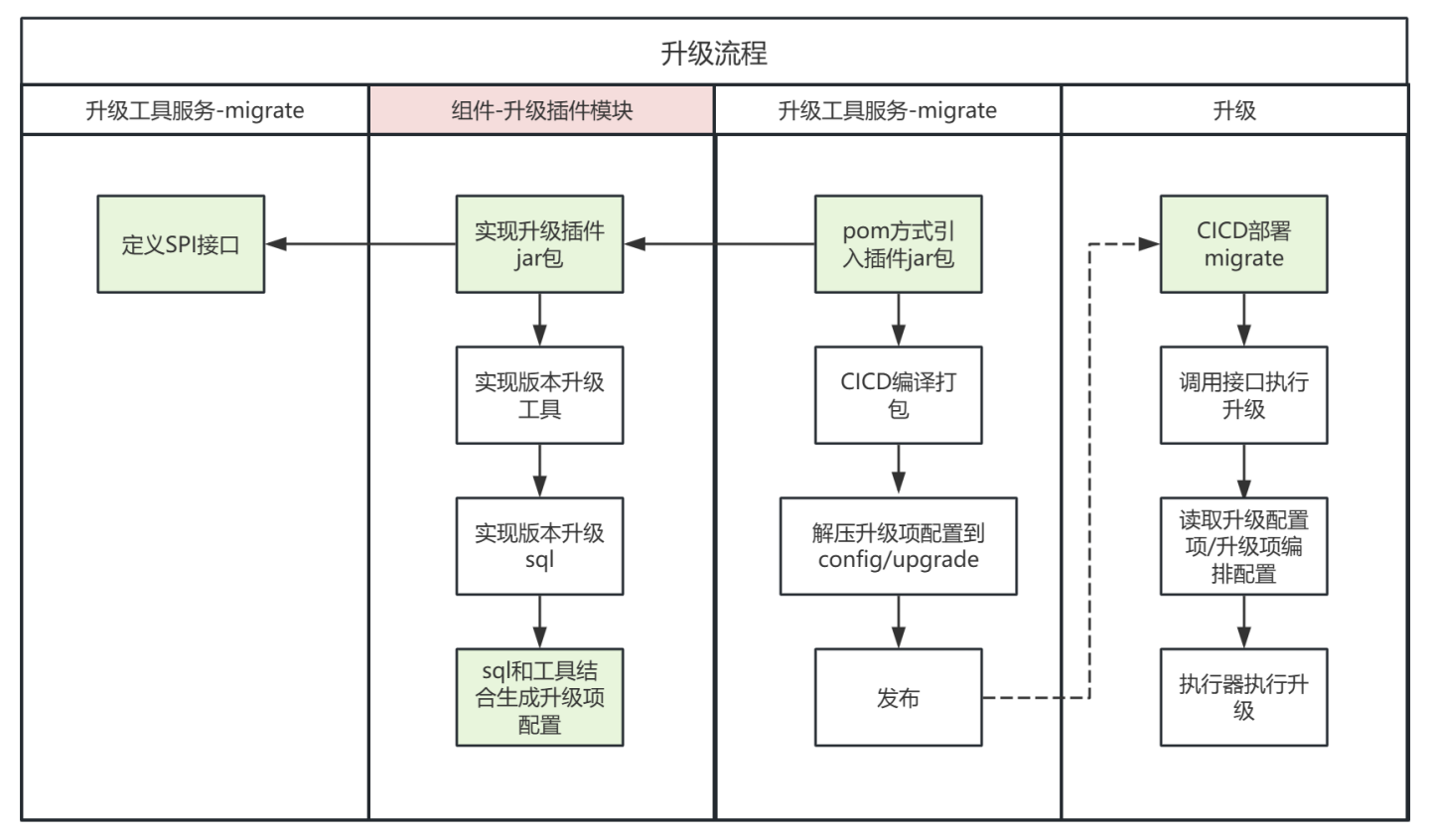
升级工具服务定义SPI接口sdk，各个组件单独创建一个modul实现升级插件，升级插件手动deploy发布到仓库；

升级工具服务引入各组件提供的升级插件；

升级工具打包的时候解压升级插件的中升级项配置文件，并打包到发布的压缩包中，方便升级出问题时，临时调整sql和执行阶段；

升级项配置文件解释：每个版本一个升级配置项文件，文件包含sql和工具方法的入口信息，执行升级的时候按顺序执行sql和调用工具；

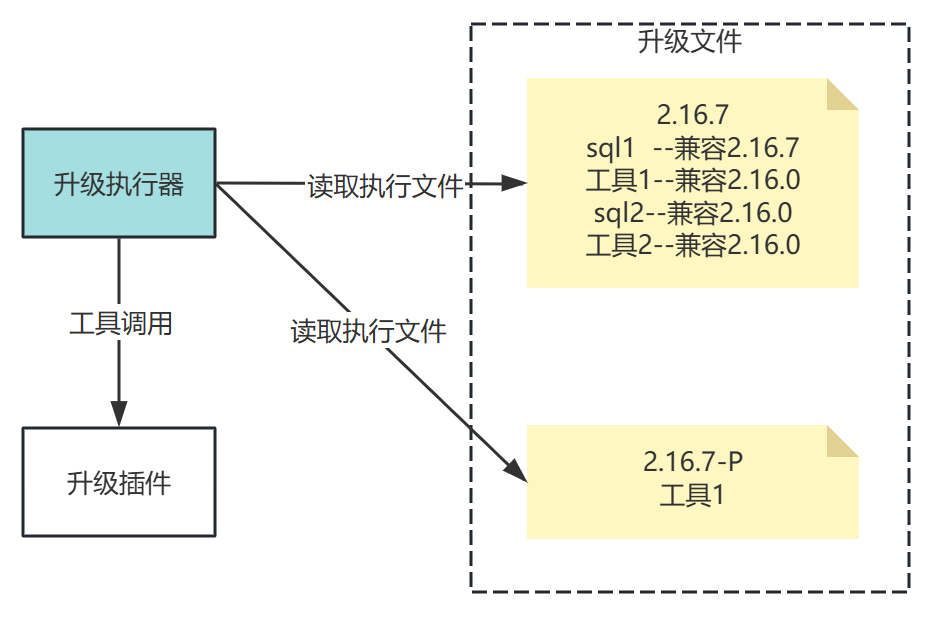
升级时，通过CICD部署升级工具，CICD在停服前、停服中、服务启动后等切点调用升级工具接口；



### 关键模块设计

#### 升级执行器

升级执行器是整个升级框架的核心组件，负责根据输入的源版本和目标版本，从升级配置中读取需要执行的SQL和工具，并按照编排配置执行相应的任务。它确保在升级过程中各项操作的顺序和依赖关系得到满足。



##### 组件结构

升级执行器的主要组件包括：

1. 版本管理模块：负责版本的兼容性检查。
2. 任务获取模块：从配置中获取需要执行的SQL和工具。
3. 编排管理模块：管理任务的执行顺序和阶段。
4. 执行管理模块：负责实际执行任务并记录结果。
5. 日志与报告模块：记录执行过程中的日志，并生成最终报告。

##### 详细流程

##### 版本管理

功能：检查源版本和目标版本的兼容性。

实现：

从配置中读取支持的版本范围。

比较输入的源版本和目标版本，确保它们在支持的范围内。

##### 任务获取

功能：根据版本信息获取需要执行的SQL和工具。

实现：

遍历升级配置，筛选出与当前版本相关的任务。

##### 编排管理

功能：管理任务的执行顺序和阶段。

实现：

根据编排配置确定任务的执行顺序。

支持停服前、停服中、服务启动后等不同阶段的任务。

##### 执行管理

功能：执行任务并记录结果。

实现：

逐个执行任务，捕获执行结果（成功、失败、跳过）。

记录每个任务的执行时间和状态。

##### 日志与报告

功能：记录执行过程中的日志，并生成最终报告。

实现：

记录每个任务的执行状态。

生成执行摘要，输出总耗时和各阶段耗时。

##### 配置示例

###### 升级项配置（upgrade\_tasks\_V2.16.7.properties）

sql.{uid}.id: 每条SQL的唯一标识符，用于在执行时引用。

sql.{uid}.description: 对SQL操作的简要描述，便于理解其功能。

sql.{uid}.file: SQL脚本的文件名，执行时需要读取该文件。

sql.{uid}.stage: 指定该SQL在升级过程中的执行阶段（如pre\_stop、in\_stop、post\_start）。

tool.{uid}.id: 每个工具的唯一标识符，用于在执行时引用。

tool.{uid}.description: 对工具操作的简要描述，便于理解其功能。

tool.{uid}.package: 工具方法的包名，执行时需要调用该方法。

tool.{uid}.stage: 指定该工具在升级过程中的执行阶段。

# SQL配置

sql.{uid}.description=备份旧数据

sql.{uid}.file={sql}

sql.{uid}.stage=pre\_stop

sql.{uid}.description=修改订单表结构

sql.{uid}.file={sql}

sql.{uid}.stage=in\_stop

sql.{uid}.description=重建索引

sql.{uid}.file={sql}

sql.{uid}.stage=post\_start

# 工具配置

tool.{uid}.description=验证数据库模式

tool.{uid}.package=com.example.tools.ValidateSchemas

tool.{uid}.stage=pre\_stop

tool.{uid}.description=清理临时数据

tool.{uid}.package=com.example.tools.CleanTempData

tool.{uid}.stage=in\_stop

###### 升级项编排配置（upgrade\_orchestration\_V2.16.7.properties）

sql.backup\_legacy\_data.stage: 指定backup\_legacy\_data SQL的执行阶段，优先于主配置文件中的阶段。

tool.validate\_schemas.stage: 指定validate\_schemas工具的执行阶段，确保在升级前执行。

# 升级项编排配置

# SQL执行阶段配置

sql.{uid}.stage=pre\_stop

sql.{uid}.stage=in\_stop

sql.{uid}.stage=post\_start

# 工具执行阶段配置

tool.{uid}.stage=pre\_stop

tool.{uid}.stage=in\_stop

#### 数据源管理

数据源管理模块负责管理不同类型的数据源，包括MySQL、ClickHouse（CK）和Redis。每个组件可以单独配置数据源，以支持多个数据实例的部署需求

##### 数据源配置示例

# MySQL来源数据源配置

spring.datasource.src.mysql.alarm.name=srcAlarmMysql

spring.datasource.src.mysql.server.name=srcServerMysql

# MySQL目标数据源配置

spring.datasource.dst.mysql.alarm.name=dstAlarmMysql

spring.datasource.dst.mysql.server.name=dstServerMysql

# ClickHouse来源数据源配置

spring.datasource.src.ck.server.name=srcServerCK

# Redis来源数据源配置

spring.datasource.src.redis.template.name=serverRedisTemplate

#### 升级插件module

##### SPI接口定义

包括资源需求检查、数据质量检查、获取兼容信息、升级耗时评估、数据备份清理等接口；

###### 资源需求检查

package com.example.upgrade.spi;

/\*\*

\* 资源需求检查接口

\*/

public interface ResourceCheck {

/\*\*

\* 检查资源需求

\* @param currentVersion 当前版本

\* @param targetVersion 目标版本

\* @param envInfo 环境信息

\* @return 资源需求检查结果

\*/

List<ResourceCheckResult> checkResources(String currentVersion, String targetVersion, EnvInfo envInfo);

}

/\*\*

\* 资源需求检查结果

\*/

public class ResourceCheckResult {

private String type; // 备份类型（如：增量备份、全量备份、数据双写）

private String name; // 业务类型（如：用户数据、订单数据）

private long value; // 备份需要的空间

private String unit; // 单位（如：GB、MB）

}

###### 数据质量检查

package com.example.upgrade.spi;

/\*\*

\* 数据质量检查接口

\*/

public interface DataQualityCheck {

/\*\*

\* 执行数据质量检查

\* @param currentVersion 当前版本

\* @param targetVersion 目标版本

\* @param envInfo 环境信息

\* @return 数据质量检查结果

\*/

DataQualityCheckResult checkDataQuality(String currentVersion, String targetVersion, EnvInfo envInfo);

}

/\*\*

\* 数据质量检查结果

\*/

public class DataQualityCheckResult {

private boolean isValid; // 检查是否通过

private String status; // 状态（通过、警告、失败）

private List<DataQualityIssue> issues; // 检查项问题列表

// Getters and Setters

}

/\*\*

\* 数据质量检查项问题

\*/

class DataQualityIssue {

private String checkItem; // 检查项名称

private String result; // 检查结果

private String suggestion; // 建议修复措施

}

###### 获取兼容信息

package com.example.upgrade.spi;

/\*\*

\* 兼容信息获取接口

\*/

public interface CompatibilityCheck {

/\*\*

\* 获取兼容信息

\* @param currentVersion 当前版本

\* @param targetVersion 目标版本

\* @param envInfo 环境信息

\* @return 兼容性检查结果

\*/

CompatibilityCheckResult getCompatibilityInfo(String currentVersion, String targetVersion, EnvInfo envInfo);

}

/\*\*

\* 兼容性检查结果

\*/

public class CompatibilityCheckResult {

private boolean isCompatible; // 兼容性状态

private String message; // 兼容性说明

private List<CompatibilityIssue> issues; // 兼容性问题列表

// Getters and Setters

}

/\*\*

\* 兼容性问题

\*/

class CompatibilityIssue {

private String taskName; // 任务名称（SQL或工具）

private String compatibilityStatus; // 兼容性状态

private String impact; // 影响描述

private String suggestion; // 建议

}

###### 升级耗时评估

package com.example.upgrade.spi;

/\*\*

\* 升级耗时评估接口

\*/

public interface UpgradeTimeEstimation {

/\*\*

\* 评估升级耗时

\* @param currentVersion 当前版本

\* @param targetVersion 目标版本

\* @param envInfo 环境信息

\* @return 耗时评估结果

\*/

TimeEstimationResult estimateUpgradeTime(String currentVersion, String targetVersion, EnvInfo envInfo);

}

/\*\*

\* 耗时评估结果

\*/

public class TimeEstimationResult {

private String taskName; // 任务名称

private long estimatedTime; // 以秒为单位的预估耗时

private String stage; // 执行阶段（停服前、停服中、服务启动后）

// Getters and Setters

}

###### 备份清理

/\*\*

\* 数据备份清理接口

\*/

public interface BackupCleanup {

/\*\*

\* 执行数据备份清理

\* @return 清理结果

\*/

CleanupResult cleanupBackup();

}

/\*\*

\* 清理结果

\*/

public class CleanupResult {

private boolean isSuccess; // 清理是否成功

private String message; // 清理结果说明

// Getters and Setters

}

###### 环境信息类

package com.example.upgrade.spi;

/\*\*

\* 环境信息

\*/

public class EnvInfo {

private String clusterId; // 集群ID

private String environmentType; // 环境类型（如测试、生产）

// Getters and Setters

}

##### 目录结构

upgrade-plugin/

│

├── src/

│ ├── main/

│ │ ├── java/

│ │ │ └── com/

│ │ │ └── example/

│ │ │ ├── upgrade/

│ │ │ │ ├── UpgradePlugin.java # 插件主类

│ │ │ │ ├── UpgradeService.java # 升级服务逻辑

│ │ │ │ ├── spi/ # SPI接口实现

│ │ │ │ │ ├── ResourceCheckImpl.java # 资源需求检查实现

│ │ │ │ │ ├── DataQualityCheckImpl.java # 数据质量检查实现

│ │ │ │ │ ├── CompatibilityCheckImpl.java # 兼容性检查实现

│ │ │ │ │ └── UpgradeTimeEstimationImpl.java # 升级耗时评估实现

│ │ │ │ ├── model/ # 数据模型

│ │ │ │ │ ├── ResourceCheckResult.java # 资源检查结果模型

│ │ │ │ │ ├── DataQualityCheckResult.java # 数据质量检查结果模型

│ │ │ │ │ ├── CompatibilityCheckResult.java # 兼容性检查结果模型

│ │ │ │ │ ├── TimeEstimationResult.java # 耗时评估结果模型

│ │ │ │ │ └── CleanupResult.java # 清理结果模型

│ │ │ ├── common/ # 公共工具类

│ │ │ │ └── util/ # 工具类包

│ │ │ │ ├── LoggerUtil.java # 日志工具类

│ │ │ │ ├── DatabaseUtil.java # 数据库工具类

│ │ │ │ └── UpgradeConfigLoader.java # 加载配置的类

│ │ │ └── upgrade\_tools/ # 升级工具目录

│ │ │ ├── V2.16.6/

│ │ │ │ ├── DataMigrationTool.java # 数据迁移工具实现

│ │ │ │ └── DataCleanupTool.java # 数据清理工具实现

│ │ │ ├── V2.16.7/

│ │ │ │ ├── DataMigrationTool.java # 数据迁移工具实现

│ │ │ │ └── DataCleanupTool.java # 数据清理工具实现

│ │ │ └── V2.16.8/

│ │ │ ├── DataMigrationTool.java # 数据迁移工具实现

│ │ │ └── DataCleanupTool.java # 数据清理工具实现

│ │ └── upgrade\_resources/ # 升级项配置目录

│ │ ├── upgrade\_tasks\_V2.16.6.properties # 升级任务配置

│ │ ├── upgrade\_tasks\_V2.16.7.properties # 升级任务配置

│ │ └── upgrade\_tasks\_V2.16.8.properties # 升级任务配置

│ └── test/

│ ├── java/

│ │ └── com/

│ │ └── example/

│ │ └── upgrade/

│ │ ├── UpgradePluginTest.java # 插件单元测试

│ │ └── UpgradeServiceTest.java # 升级服务测试

│ └── resources/

│ └── test\_datasource.properties # 测试用数据源配置

│

└── pom.xml # Maven配置文件

##### 升级工具调用

在升级配置项的Properties文件中，添加调用工具的包路径。升级执行器将根据这些路径动态加载和执行相应的升级工具。这种设计提高了系统的灵活性和可扩展性，允许在不同版本间轻松切换和管理工具。

所有升级工具需实现统一的接口，以确保执行的一致性

package com.example.upgrade.tools;

public interface UpgradeTool {

void execute(); // 执行工具的主逻辑

}

##### 生成升级项配置（sql+工具进行编排）

##### 升级插件deploy

手动deploy到仓库

#### 对外接口

1. **升级前评估**
   1. 包括：资源需求检查、数据质量检查、获取兼容信息、升级耗时评估
   2. 输入
      1. currentVersion，升级开始版本，示例：V2.16.6
      2. tagVersion，升级结束版本，示例：V2.16.7，或V2.16.7-P1-T1
      3. src数据源、dst数据源
      4. eninfo，局点环境信息
   3. 输出
      1. 资源需求检查结果；
      2. 数据质量检查结果;
      3. 是否兼容结果；
      4. 耗时评估结果；
   4. 异常
      1. 执行失败中断执行，人工处理异常；
2. **数据升级**
   1. 数据升级包括sql脚本执行、数据升级工具执行、通用工具执行，执行阶段有停服前、停服中、服务启动后；
   2. 输入
      1. currentVersion，升级开始版本，示例：V2.16.6
      2. tagVersion，升级结束版本，示例：V2.16.7，或V2.16.7-P1-T1
      3. src数据源、dst数据源
      4. 执行阶段（停服前、停服中、服务启动后）
      5. eninfo，局点环境信息
      6. 升级服务名
   3. 输出
      1. 执行结果日志；
   4. 异常
      1. 执行失败中断执行，人工处理异常；

#### 服务优雅停止/启动

##### 服务停止前从nacos下线

停止服务时，服务先从nacos中下线，避免nacos-client主动发现节点下线延迟，把请求路由到停止的服务节点；

CICD停止服务前，先把服务从nacos下线，等待5秒（nacos-client配置的发现间隔），然后再停止服务；

##### 服务停止时异步任务优雅结束

1. 异步任务想要优雅退出，必须先说明下linux kill命令：

* kill：默认发送SIGTERM信号，进程可以捕获并执行清理操作。
* kill -9：发送SIGKILL信号，进程无法捕获，立即终止进程，不会执行清理操作。

1. CICD在停止进程时，建议先使用kill（即SIGTERM）来尝试正常终止进程。如果15秒进程未退出，再使用kill -9（即SIGKILL）强制终止进程。
2. 对于耗时较长的异步任务，需要识别程序中断是否影响，对于影响业务的需要优雅退出；或重启后检测异步任务是否状态是否正常；

##### 服务停止时Kafka消费正常退出

这里只列举java怎么正常退出kafka消费者

###### kafka手动消费者退出

@Autowired

private KafkaConsumer<String, String> consumer;

@Bean

public KafkaConsumer<String, String> kafkaConsumer() {

Properties props = new Properties();

props.put(ConsumerConfig.BOOTSTRAP\_SERVERS\_CONFIG, bootstrapServers);

props.put(ConsumerConfig.GROUP\_ID\_CONFIG, groupId);

props.put(ConsumerConfig.KEY\_DESERIALIZER\_CLASS\_CONFIG,

……

consumer = new KafkaConsumer<>(props);

consumer.subscribe(Collections.singletonList("test-topic"));

return consumer;

}

@PreDestroy

public void destroy() {

if (consumer != null) {

consumer.close();

}

}

###### @KafkaListener消费者退出

@Autowired

private MessageListenerContainer listenerContainer;

@KafkaListener(topicPattern = KafkaTopicConstant.Config.BASE\_CONFIG\_EVENT\_LINKAGE,

containerFactory = "kafkaListenerContainerFactoryBatch",

id = KafkaTopicConstant.Config.BASE\_CONFIG\_EVENT\_LINKAGE,

idIsGroup = false)

public void process(List<String> list, Acknowledgment ack) {

……

}

@PreDestroy

public void destroy() {

MessageListenerContainer listenerContainer = kafkaListenerEndpointRegistry.getListenerContainer(KafkaTopicConstant.Config.BASE\_CONFIG\_EVENT\_LINKAGE);

if (listenerContainer != null) {

listenerContainer.stop();

}

}

##### 服务启动速度和可用状态

1. 对于服务启动需要保障能正常提供业务后才能注册到nacos中；如中台最后1条gps需要加载全量最后1条gps到内存，未加载完成，用户调用会返回脏数据导致业务不正常；
2. 原则异步加载的内容不能影响正常业务调用，如果影响业务请调整为注册nacos前加载；
3. 10W设备接入，服务启动控制在2分钟内，服务启动速度优化方法：
   1. 精简依赖：pom.xml移除不必要的库和依赖；
   2. 异步初始化：对于一些可以异步完成的任务，在不影响主流程的情况下，考虑将其放到后台线程中执行；
   3. 代码层面：检查静态块和静态变量的使用，确保它们不会执行耗时操作;
   4. 代码层面：对于Spring等框架的应用，检查是否有过多的Bean定义，考虑使用条件注解（如@Conditional）来控制Bean的创建；
   5. 减少类加载：排查并延迟非必要的组件初始化，直到它们真正被需要时再加载；使用懒加载模式来优化对象创建时机;
4. 中台启动异步加载内容，影响正常业务清单：待各模块负责人补充；

## **验证方案**

## **功能测试**