GreatDB
万里数据库

用户手册 2025

万里数据库评估迁移工具软件

稳定 · 性能 · 易用

北京万里开源软件有限公司
Beijing Great OpenSource Software Co., Ltd.

目录

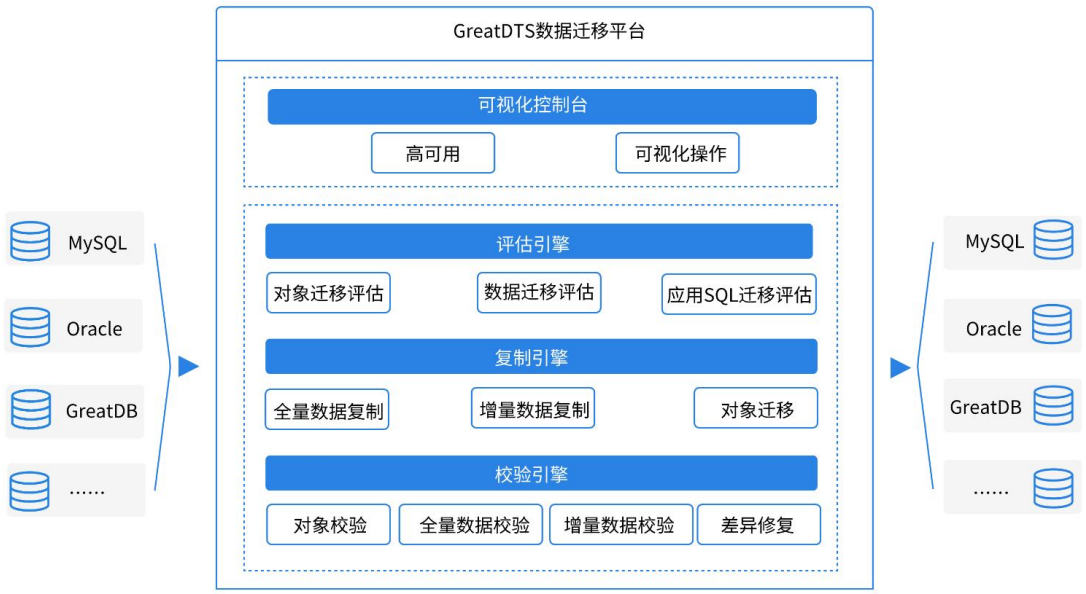
1. GreatDTS 简介

GreatDTS 是集评估、迁移、同步和校验功能于一体的数据迁移工具，提供便捷部署、易于使用、快速高效的同构和异构数据复制服务。

GreatDTS 支持对象迁移评估、应用改造评估、全量数据迁移、增量数据同步和数据校验等功能，具备应用深度分析、语法树分析、断点续传、多线程多任务并行模式、事务级同步和全量数据一致性校验等特性。支持用户迁移 Oracle 数据库到 GreatDB 系列数据库，同时提供 GreatDB、Oracle、MySQL、ClickHouse、Kafka 等多种数据源之间的数据同步。

1.1 整体架构

GreatDTS 提供了一套分布式、易扩展、高可靠的产品架构，产品功能架构图如下



整个产品功能架构分为三部分：

- 可视化控制台：用户交互层，可视化控制台提供数据管理的任务创建、管理、运维、安全操作审计等。
- 功能引擎层：负责具体的管理工作。包含评估引擎、迁移引擎、校验引擎
- 数据源连接层：负责为上层引擎提供数据源的连接访问。

1.2 平台支持

如下为 GreatDTS 支持的 CPU 和操作系统，其中 Oracle 增量复制组件仅支持 x86_64 架构。

CPU 类型	OS 兼容列表
x86_64	Redhat/CentOS 7.4 及以上版本
	OpenEuler 20.03 以上版本
	Ubuntu 18 LTS 及以上版本
	统信服务器操作系统(UOS) V20
	麒麟操作系统 V10
	拓林思企业级服务器操作系统 V15.0 & V16.0
	中移统 BC-Linux V21.1
arm64	Redhat/CentOS 7.4 及以上版本
	OpenEuler 20.03 以上版本
	Ubuntu 18 LTS 及以上版本
	统信服务器操作系统(UOS) V20
	麒麟操作系统 V10
	拓林思企业级服务器操作系统 V15.0 & V16.0
	中移 BC-Linux V21.1

1.3 功能概览

源数据库	**目标数据库**	迁移评估	结构迁移	全量复制	增量复制	结构对比	数据对比
GreatDB	GreatDB			✓	✓	✓	✓
	MySQL			✓	✓	✓	✓

	Oracle			✓	✓	✓	✓
GreatDB Cluster	GreatDB Cluster			✓	✓	✓	✓
	GreatDB			✓	✓	✓	✓
	MySQL			✓	✓	✓	✓
	Oracle			✓	✓	✓	✓
MySQL5.7+	GreatDB	✓	✓	✓	✓	✓	✓
	MySQL	✓	✓	✓	✓	✓	✓
	Oracle			✓	✓	✓	✓
Oracle 11G +	GreatDB	✓	✓	✓	✓	✓	✓
	MySQL	✓	✓	✓	✓	✓	✓
	Oracle			✓	✓	✓	✓
ClickHouse v24.1.2.5-stable 及以上	GreatDB		✓	✓			
	MySQL		✓	✓			
MyCat1.x、2.x	GreatDB Cluster			✓	✓		
	GreatDB			✓	✓		

2. 核心原理

2.1 高可用 HA

GreatDTS 迁移服务（Great Data Transmission Service，DTS）的高可用模块（High Available，简称 HA）为数据传输链路的稳定性和连续性提供保障。当 DTS 部署服务器发生单点故障或个别任务进程发生异常时，HA 能够及时发现、介入并尝试恢复。

2.1.1 HA 组件维护

组件名称	用途	是否具备高可用能力
master	管理控制台	是
worker	任务运行进程	否

2.1.2 HA 触发场景

	master 管理控制台	Full-replication 全量复制	incr-replication 增量复制	Structure migration 结构迁移	comparison 数据对比
组件异常/任务异常	✓	✓	✓	✗	✗
机器宕机	✓	✗	✗	✗	✗

1. 机器宕机

当前版本 Master 主节点宕机时，会重新选举产生新的 Master 主节点；Worker 节点会向新的 Master 汇报任务状态。

2. 组件异常/任务异常

全量复制/增量复制可通过配置失败恢复策略来处理异常；发生异常时整个任务会被重新启动。故障恢复策略如下：

- none / off / disable ：不启用故障恢复策略
- Fixed-delay：固定延迟的失败恢复。发生故障后，在延迟一段时间后，重启任务。
- Failure-rate：固定时间内的最大失败率。如 5min 内，最多允许 3 次失败；3 次以内故障，任务会被重启，超过此数判定为失败。

3. master 高可用实现说明

首先自动切换无需人工干预，其次 master 的部署个数等于 2 即可，无多数派限制

初始化后首次选主，第一个 worker 第一次连接配置文件中的任意一个 master，此时发起 master 的选主，被链接的 master 发起分布式锁，查询当前的主 master，没有发现

master 时，将自己设置为主，与当前 worker 建立连接，如已有主 master 就向主 master 发起心跳，心跳正常则 worke 与主 master 建立连接。

主 master 异常，选主期间 worker 是重试状态，但是相关的任务是正常的。除非所有 master 都异常了，worker 才会断开。worker 断开才会影响任务。

从新选主后，老主 master 的缓存中实时点位会丢失，新 master 启动后会从元数据库中取断点信息，重复执行断点和最新的点位的数据。如下为断点续传的在每个阶段的实现情况。

a. 全量阶段：
   i. 有主键表，按主键更新，重复执行无影响
   ii. 无主键表，无法断点续传，需要重新做全量
b. 增量阶段
   i. 有主键表，按主键更新，重复执行无影响
   ii. 无主键表，按事务号更新，重复执行无影响（事务号 被记录在目标库中 dts_tx 表）

4. worker 高可用实现说明

worker 是无状态的，同一个任务目前只支持一个 worker，当前 worker 挂了，任务重启后会在健康的 worker 中选择一个继续执行任务，点位信息在 master 中记录不涉及数据丢失的问题。

2.2 CDC-Connector 原理介绍

DTS 的 CDC 是通过 CDC-Connector 组件实现的。常见的 CDC 技术有 Trigger-based CDC（基于触发器的实现）、Timestamp-based CDC（基于时间戳的实现）和 Log-Based CDC（基于日志的实现）。DTS 的 CDC-Connector 组件是 Log-Based 的 CDC，相较于其它方式，对数据库的侵入和影响更小，实时性更高。

CDC-Connector 拉取并解析数据库日志进行拉取解析，并对解析后的日志记录按统一格式存储。下游可以从 CDC-Connector 中以一致的方式消费增量数据，无需关心不同数据库间日志格式的差异。

CDC-Connector 负责拉取和解析各个数据库的日志。当前具体的实现 oracle-cdc-connector 和 mysql-cdc-connector。

2.2.1 Oracle-CDC-Connector

oracle-cdc-connector 有两种解析方式：

- Oracle LogMiner：Oracle 原生的日志解析

- GreatDTS-Oracle-LogMiner：DTS 自研 Oracle 日志挖掘

两种实现的区别是，原生的 Oracle LogMiner 通过 Oracle Database Server 解析 archive log；受限于实现逻辑和有限的资源占用逻辑，性能低；GreatDTS-Oracle-LogMiner 直接读取和解析 archive log 文件，性能高

2.2.2 MySQL-CDC-Connector

MySQL-CDC-Connector 主要流程包括：日志拉取、Binlog Event 预解析过滤 和 Binlog Event Value 解析

- 日志拉取
  - 模拟成 MySQL 数据源的一个从库接收源端的增量二进制日志记录（与源端建立 Binlog Dump 连接，接收其发送过来的二进制日志流），并根据 Event 的长度将日志流拆分成完整且独立的 Binlog Event。
- Binlog Event 预解析过滤
  - 对 Binlog Event 进行预解析，根据黑白名单进行过滤，仅保留需要同步的表的 Binlog Event。
- Binlog Event Value 解析
  - 根据列类型对列值做解析。

2.3 数据传输原理

在 DTS 数据传输场景中（包括数据迁移和数据同步功能），将数据源从源端到目标端的复制分为两个阶段：全量数据复制和增量数据复制。从数据流动的视角来看，两者都是从源端获取数据，同步到目标端。在 DTS 中，负责全量数据复制的组件为 Full-Replication，负责增量数据复制的组件为 Incr-Replication。

Full-Replication/Incr-Replication 运行模式如下：

1. Full-Replication/Incr-Replication 被划分为三个大模块，分别是 Source、Transformer 和 Sink。
2. Source 负责从源端抽取数据并将数据发送给 Transformer。
3. Transformer 负责进行 ETL 等数据处理，然后将处理后的数据发送给 Sink。
4. Sink 负责将数据写入到目标端。

Full-Replication/Incr-Replication 使用 Record 作为数据流转的统一模型，在全量数据读取中 Record 表示源端的一行数据，在增量数据读取中 Record 表示源端的一行数据变更。当源端一条 SQL 产生了 N 条数据变更时（即 affected rows 为

N），Source 会读取到 N 个 Record。

Record 中包含的主要内容如下：

- 结构信息，包括：
  - 库名、表名。
  - 列信息，包括列名、列类型和精度等。
  - 主键/唯一键。
- 数据信息，主要是每一列的值。
- 操作类型，例如 Insert、Update、Delete 和 DDL 等。
- 变更发生事件，只存在增量数据复制中。

在介绍了 Full-Replication/Incr-Replication 的运行模式之后，接下来为您介绍 Full-Replication/Incr-Replication 的工作原理。包含以下内容：

1. DTS 针对不同源端和目标端是如何做数据的读取和写入的。
2. DTS 如何确保数据复制过程中源端和目的端数据是一致的。
3. DTS 如何保证数据复制的效率，包括复制全量数据和复制增量数据

2.3.1 全量复制

DTS 读取全量数据读取包括以下四个步骤：

- 根据配置信息（可能通过某些规则）确定源端需迁移的库表。
- 查询这些库表的元信息，主要包括主键（或非空唯一键）以及列信息等。
- 分析源端表的特征，选择适当的数据切片方法进行数据分片。
- 根据每个切片读取数据，并将读取到的数据发送出去。

1. 数据读取 - JDBC Source

JDBC Source 负责全量迁移的数据读取任务。数据读取分为三个流程：

- **元信息查询**：查询需要同步的库表元信息，主要读取的信息有 Table 的 Schema 信息、索引信息和分区信息。比如 MySQL 读取的统计库就是 `information_schema` 中的统计表信息。
- **切片**：假设表数据是以 `(c1, c2)` 作为索引列，以索引作为排序可以得到 `[(c1_start, c2_start), (c1_end, c2_end)]` 的主键数据序列。在这数据序列中以特定大小将序列拆成 `(null, (c1_start_1, c2_start_1)]`，`((c1_start_1, c2_start_1), (c1_start_2, c2_start_2)]`...`((c1_start_n, c2_start_n), (c1_end, c2_end)]`，`(c1_end, c2_end]` 的序列切片操作，每一个序列为一个切片。

- **查询数据**：根据 `((c1_start, c2_start), (c1_end, c2_end)]` 这样的一个切片我们可以生成对应查询数据的 SQL。再结合内存限流等机制生成一批 RecordBatch 流入到 Full-Replication 框架中。生成 RecordBatch 的过程即将数据库特有的数据类型转换成 Java 统一的数据类型。

2. 数据写入 - JDBC Sink

JDBC Sink 主要负责数据的写入，其中全量写入较增量简单。数据写入分为以下步骤：

- **RecordBatch 重组**：在部分场景中，比如写入分区场景中可能会对相同分区的消息进行聚合。
- **目的端 Schema 缓存**：主要针对异构数据库，我们需要根据得到的 Record Value 类型与目的端 Schema 进行 `n * m` 的类型匹配，将 Java 类型转换成数据库可以写入的类型。
- **数据写入**：以 RecordBatch 为组，构建 Batch 的语句，以事务的方式写入到目标端。
- **写入方式**：DTS 为用户提供多种写入方式以适配不同场景，如有主键/唯一键的表可以选择使用 INSERT 或者 MERGE 等 4 种方式

3. 全量迁移断点续传

在 Sink 写入成功之后由框架层来通知 Source 写入成功，Source 根据写入成功的信息维护最小的切片点。在切片的说明中可以推出，以相同 Snapshot 场景来说形成的序列顺序是相同的，有了最小切片位点实际 Full-Replicaiton 可以知道可以从哪个特定数据序列开始进行切片操作。

2.3.2 增量复制

1. 数据读取 - CaptureChangeSource

CaptureChangeSource 主要获取 cdc-connector 传递来的消息，然后将消息转换成框架适配的 RecordBatch。在转换过程中，我们主要做了以下一些工作：

- **事务性消息**：对于无主键表的变更消息会随着的 `begin` 和 `commit` 中的组成一个 RecordBatch 传递到框架中。
- **数据合并消息**：对于有主键/唯一键的表的消息，会直接破坏跨原事务投递（最终一致）

2. JDBC-Sink 对于事件的针对性优化

对于数据库中一行数据（唯一行标记）可能会由于多次变更（如 `INSERT -> UPDATE -> DELETE`），从而在短时间内出现多条变更消息。JDBC-Sink 针对这种场景会将多条

事件合并成最终结果事件，只向目标库投递最终结果的事件，以提高写入效率。

2.3.3 最终一致性

DTS 为了保障复制效率，不是按事务做的回放，而是一个事务会被拆成多个线程并发的去回放，这个事务中所有的并发都提交了才会一致，所以是最终一致性。

2.4 数据校验原理

本文介绍 DTS 数据校验原理。通过将数据切片，并结合缓存、Join 以及校验过程，DTS 能够识别和处理不一致数据。

名词解释

名称	描述
切片	切片是一种数据库优化策略，它通过将大表分解为更小的部分来提高并发查询的效率。例如，一张包含 300 条记录的表（假设主键为 id，值从 1 到 300）被分成了 3 个分片： 切片 1：id < 100 切片 2：101 <= id <= 200 切片 3：id > 200 每个切片代表了表的一个子集。切片适用于大规模的数据库系统，在这些系统中，整张表的全表查询会非常耗时。通过将大型表进行分割，数据库的性能和可扩展性可以得到提升。
读数据	通过 JDBC 对源端和目标端执行带有特定查询条件的数据检索操作。
缓存	缓存是指源端和目标端读出数据后，无主键表的两端的数据，由于默认的排序规则不同，返回的结果不同，首次校验不一致的数据会先进入的缓存，等待二次校验。
校验	从 数据流 中拿出行数据，对数据的全字段进行对比。
复检	首次校验出来不一致的数据后，数据会先进入到缓存。复检是对缓存中的数据做二次校验。
订正	校验出来不一致的数据后，根据不一致的信息和该记录的主键/唯一键直接生成一条 INSERT/UPDATE/DELETE SQL 执行到目标端，保持该主键/唯一键对应的记录和源端数据一致。

切片线程	多个切片线程可以并发实现多张表的切片功能。
数据源读线程	分源端读线程和目标端读线程，分别可以并发地从源端/目标端读数据。
校验线程	多个校验线程可以并发校验数据。
订正线程	多个订正线程可以并发地向目标端订正数据。管理控制台 操作执行

2.4.1 数据校验流程

DTS 按照以下流程进行全量数据校验：

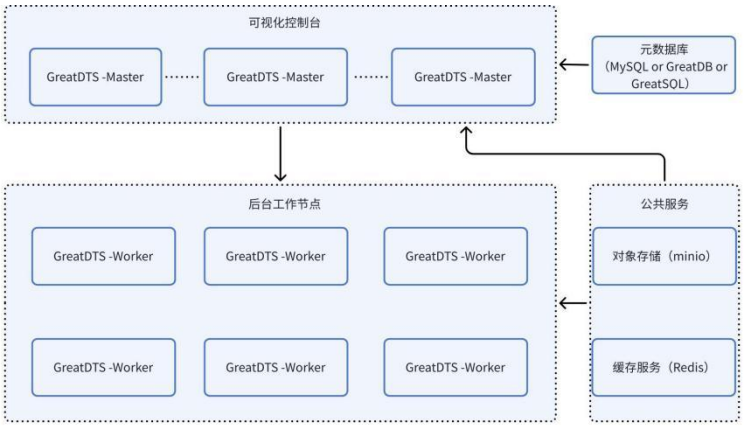
1. 切片线程对源端表切片，完成后的切片都会放入队列 1 中。
2. 源端和目标端的数据源读线程从队列 1 中取出切片，根据切片在源端和目标端数据库中并行读取数据。
3. 对同一个切片，从源端和目标端读出的数据，逐行进行校验，不一致的数据会先进入到缓存，待首次校验完全后，对缓存中的数据进行复检。
4. 校验线程从队列 2 中取出 数据 成功的数据进行校验。先校验数据行标示（唯一键）是否一致，再校验剩余行是否一致。不一致的行将记录信息提交给异步记录线程
5. 收集差异信息，生成差异报告、订正脚本等

2.4.2 数据校验原理

DTS 采用流式读取源和目标到本地的方式逐行逐列进行对比的方式。以消除异构库之间的精度差异。

3. 安装部署

3.1 部署架构



- GreatDTS-Master：可视化控制台，负责任务的配置、管理和调度。一般采用多节点多实例部署，多节点互备，只会有一个被选为主节点。轻负载。
- GreatDTS-Worker：后台工作节点，运行高负载任务的进程，复制、对比等任务均运行在工作节点上
- 元数据库：一般是 MySQL 或 GreatDB 或 GreatSQL，Master 的元数据和配置信息会被写入到这里，如果想使用 GreatDB 请确认已获得官方授权
- 对象存储：使用 minio 作为对象存储，用于归档 DTS 的运行日志、校验报告等内容
- 缓存服务：使用 redis 作为缓存服务，增量对比的数据被记录到缓存服务上。在使用到 oracle 增量复制时，需要额外部署 kafka

3.2 硬件要求

最低硬件要求是最小化部署一套 GreatDTS 的硬件要求

	最低硬件要求
CPU	8core，2.1GHZ
Memory	32G
Hard Disk	128G+ SSD

3.3 版本要求

GreatDTS 推荐使用最新版本，如下为 GreatDTS 适配和推荐的组件版本

组件	支持版本	推荐版本
MySQL	8.0 及以上	8.0.41
GreatDB	1.0.0.6.0.1 及以上	1.0.0.6.1.0
GreatSQL	GreatSQL-8.0.25-15 及以上	GreatSQL 8.0.32-27
minio	2024.2.17 及以上	2024.08.03
redis	6.0 及以上	7.0.15
JDK	1.8	1.8

3.4 最小化部署

最小化部署是将，将所有组件部署在一台机器上，可用于个人学习和功能测试

组件	描述	部署情况
GreatDTS- Master	可视化控制台	1
GreatDTS- Worker	后台工作节点	1
MySQL or GreatDB or GreatSQL	依赖组件元数据库	单机也可以复用已存在的数据库
minio	依赖组件对象存储	单机部署
redis	依赖组件缓存服务	单机部署

3.5 推荐部署

所有组件分别独立部署，部署的数量和硬件配置根据业务量和性能要求来定。

组件	描述	部署情况
GreatDTS- Master	可视化控制台	3
GreatDTS- Worker	后台工作节点	根据业务量和性能要求来定
MySQL or GreatDB or GreatSQL	元数据库	高可用配置
minio	对象存储	高可用配置
redis	缓存服务	高可用配置

3.6 安装部署

安装部署分为两部分，一部分是依赖组件安装，一部分是 GreatDTS 的服务安装。

依赖组件分为两种情况

情况一：新建资源，本章节针对组件的第一种情况做描述。

情况二：复用已有资源，复用已有资源需要符合“硬件要求”章节中的版本要求，在 GreatDTS 配置文件中填写已有资源的配置即可。

3.6.1 创建操作系统用户

建议新建 greatdts 并授予 sudo 权限

```
JSON
# root 操作
groupadd greatdts
useradd -g greatdts greatdts
# 接着输入并确认密码
passwd greatdts
# 将解压后的目录移到合适位置赋权给 greatdb
chown -R greatdb:greatdb

# 添加 sudo 用户 greatdts
vi /etc/sudoers
greatdts    ALL=(ALL)    NOPASSWD:ALL
```

3.6.2 目录规划

目录描述	目录
数据 home 目录 (BASE_DIR)	/greatdts
存放软件包目录	$BASE_DIR/install_tools
GreatDTS 的安装目录	$BASE_DIR/greatdts
minio 安装目录	$BASE_DIR/minio
redis 安装目录	$BASE_DIR/redis
mysql 安装目录	$BASE_DIR/mysql

3.6.3 检查 JDK 版本

使用如下版本检查 JDK 的版本，如果版本不在支持的范围内，请自行更换，注意修改环境变量（PATH）本章节不展开描述。

```toml
java -version
```

3.6.4 安装依赖

安装 docker（可选）

1. 官网下载 tar 包

https://download.docker.com/linux/static/stable/

本次演示使用的是 24.0.6

2. 解压安装包

使用以下命令解压下载的 tar.gz 包：

```Bash
tar -zxvf docker-24.0.6.tgz
```

3. 将 docker 放在指定位置

将解压后的文件复制到系统的可执行文件目录，通常是 `/usr/bin` 或

`/usr/local/bin`。

```
Bash
mv docker /usr/local/bin
```

4. 配置系统服务

创建或编辑 `/etc/systemd/system/docker.service` 文件，添加以下内容：

注意修改 ExecStart 为 dockerd 的实际目录

```
toml
[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
After=network-online.service docker.socket
Wants=network-online.service

[Service]
Type=notify
ExecStart=/usr/local/bin/docker/dockerd
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutSec=0
RestartSec=2
Restart=always
StartLimitBurst=3
StartLimitInterval=60s

[Install]
WantedBy=multi-user.target
```

5. 启动 Docker 服务

执行以下命令启动 Docker 服务，设置开机自启：

```
Bash
# 启动Docker服务
sudo systemctl start docker
# 设置开机自启
sudo systemctl enable docker
```

6. **设置环境变量**

```
toml
```

```
vi ~/.bashrc
export PATH=$PATH:/usr/local/bin/docker
source ~/.bashrc
```

7. **验证是否启动成功**

```toml
toml
docker --version
```

8. **将 greatdts 加入到 docker 用户组**

```toml
toml
# 创建 docker 组（若该组不存在）
sudo groupadd docker

# 将当前用户添加到 docker 组
sudo usermod -aG docker $USER

# 更新用户组设置，使修改立即生效
newgrp docker
```

安装 MySQL

本章节仅演示使用 docker 镜像方式安装 MySQL，使用二进制安装 MySQL 和配置 MySQL 的高可用可以通过查看 MySQL 的官方手册完成。

本次演示使用的版本是 24.0.6 docker 镜像包，可以从万里技术支持团队获取。

1. **解压 MySQL 镜像包**

```toml
toml
tar -zxvf mysql_8.0.41.img.tar.gz
```

2. **导入镜像包**

```toml
toml
docker load -i mysql_8.0.41.img
```

3. **查看导入的镜像**

```toml
toml
docker images
```

4. **配置启动 docker 镜像标红色部分记得修改**

```toml
docker run --restart=always  -d \
-p 3310:3306 \
-e "MYSQL_ROOT_PASSWORD=root" \
-v "/greatdts/dts_mysql:/var/lib/mysql" \
--name "dts_mysql"  "mysql:8.0.41"
```

5. **登陆 MySQL 执行命令**

方式一

```toml
docker exec dts_mysql mysql -uroot -proot -S /var/run/mysqld/mysqld.sock
-e "create database greatdts;"
```

方式二

```Bash
docker exec -it dts_mysql bash
bash-5.1#  mysql -uroot -proot -S /var/run/mysqld/mysqld.sock
mysql: [Warning] Using a password on the command line interface can be
insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 8.0.41 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

mysql> create database greatdts;
```

安装 Minio

本章节会演示使用 docker 镜像方式安装 minio 单机，使用二进制安装 minio。如需配置 minio 的高可用可以通过查看的官方手册完成。镜像包可以通过官方下载可以通过

万里技术支持团队获取。

使用 docker 镜像方式安装 minio 单机

1. **解压**

```
Bash
tar xf bitnami_minio_2024.2.17.img.tar.gz
```

2. **导入镜像包**

```
toml
docker load -i bitnami_minio_2024.2.17.img
```

3. **查看导入的镜像**

```
toml
docker images
```

4. **配置启动 docker 镜像**

记录好端口和用户名密码，后续链接 minio 会用到，minio 的数据库目录的权限要改为 777 如下：

```
toml
chmod 777 /greatdts/minio/data
chown -R 1001:1001 /greatdts/minio/data
```

标红色部分请根据具体配置需改

```
toml
docker run --restart=always --privileged=true -d \
-p 9000:9000 -p 9001:9001 \
-e "MINIO_ROOT_USER=admin" \
-e "MINIO_ROOT_PASSWORD=admin1234" \
-v "/greatdts/minio/data:/bitnami/minio/data" \
--name "minio" "bitnami/minio:2024.2.17"
```

> bitnami 的 minio data 地址与官方不一样（/bitnami/minio/data），需要注意！

使用二进制包方式安装，布署单节点单磁盘的 MinIO 服务

1. 获取二进制包

获取二进制包的方式有两种

1）通过官方网站下载

```
Plain Text
wget https://dl.minio.org.cn/server/minio/release/linux-amd64/minio
chmod +x minio
sudo mv minio /usr/local/bin/
```

2）使用 GreatDTS 中自带的安装包

```
Plain Text
tar -xvf  minio-binary.tar.xz
chmod +x minio
sudo mv minio /usr/local/bin/
```

2. 创建配置文件/etc/default/minio

输入以下内容（MINIO_VOLUMES 为 MinIO 的存储的位置，请根据实际自行调整）

```
Plain Text
# MINIO_ROOT_USER and MINIO_ROOT_PASSWORD sets the root account for the
MinIO server.
# This user has unrestricted permissions to perform S3 and administrative
API operations on any resource in the deployment.
# Omit to use the default values 'minioadmin:minioadmin'.
# MinIO recommends setting non-default values as a best practice,
regardless of environment

MINIO_ROOT_USER=admin
MINIO_ROOT_PASSWORD=admin123

# MINIO_VOLUMES sets the storage volume or path to use for the MinIO
server.

MINIO_VOLUMES="/data/minio"

# MINIO_OPTS sets any additional commandline options to pass to the MinIO
server.
# For example, `--console-address :9001` sets the MinIO Console listen
port
MINIO_OPTS="--address :9000 --console-address :9001"
```

3. 创建 systemd 启动脚本/usr/lib/systemd/system/minio.service

脚本中使用了 minio-user 用户，请根据实际自行调整

```
Bash
[Unit]
Description=MinIO
Documentation=https://docs.min.io
Wants=network-online.target
After=network-online.target
AssertFileIsExecutable=/usr/local/bin/minio

[Service]
Type=notify

WorkingDirectory=/usr/local

User=minio-user
Group=minio-user
ProtectProc=invisible

EnvironmentFile=-/etc/default/minio
ExecStartPre=/bin/bash -c "if [ -z \"${MINIO_VOLUMES}\" ]; then echo
\"Variable MINIO_VOLUMES not set in /etc/default/minio\"; exit 1; fi"
ExecStart=/usr/local/bin/minio server $MINIO_OPTS $MINIO_VOLUMES

# Let systemd restart this service always
Restart=always

# Specifies the maximum file descriptor number that can be opened by this
process
LimitNOFILE=65536

# Turn-off memory accounting by systemd, which is buggy.
MemoryAccounting=no

# Specifies the maximum number of threads this process can create
TasksMax=infinity

# Disable timeout logic and wait until process is stopped
#TimeoutSec=infinity
```

```
TimeoutSec=0

SendSIGKILL=no

[Install]
WantedBy=multi-user.target

# Built for ${project.name}-${project.version} (${project.name})
```

4. 创建运行用户，并给存储目录赋权

```
Shell
groupadd -r minio-user
useradd -M -r -g minio-user minio-user
mkdir -p /data/minio
chown minio-user:minio-user /data/minio
```

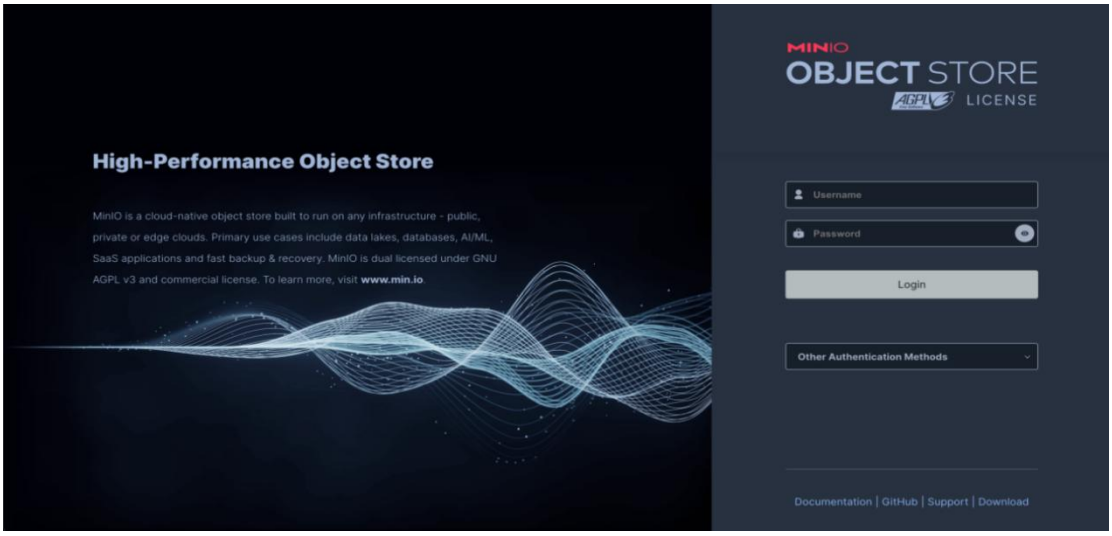
5. 启动

```
Shell
systemctl start minio
```

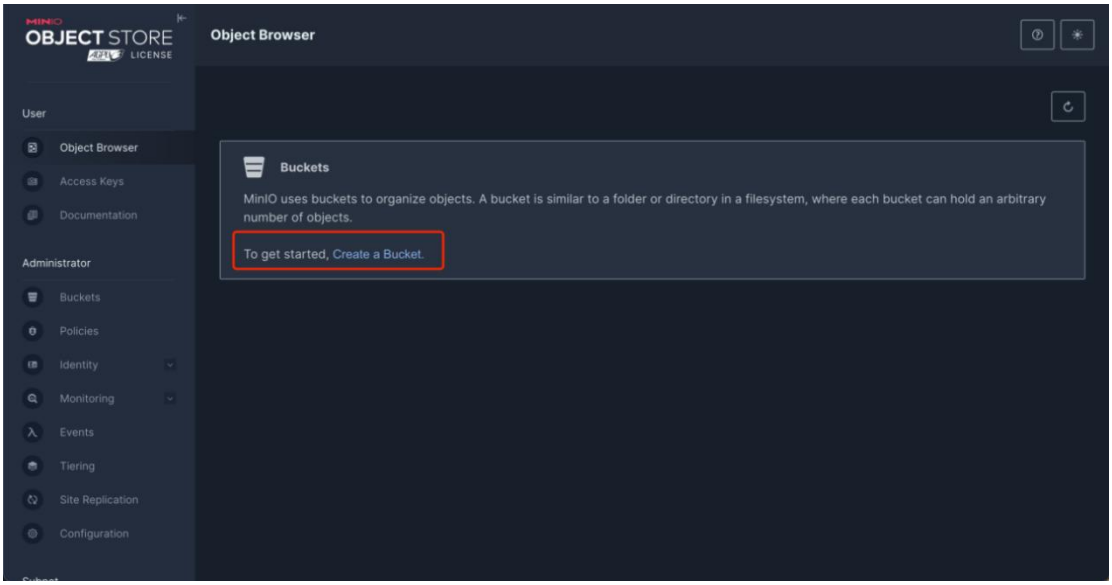
创建 minio 密钥

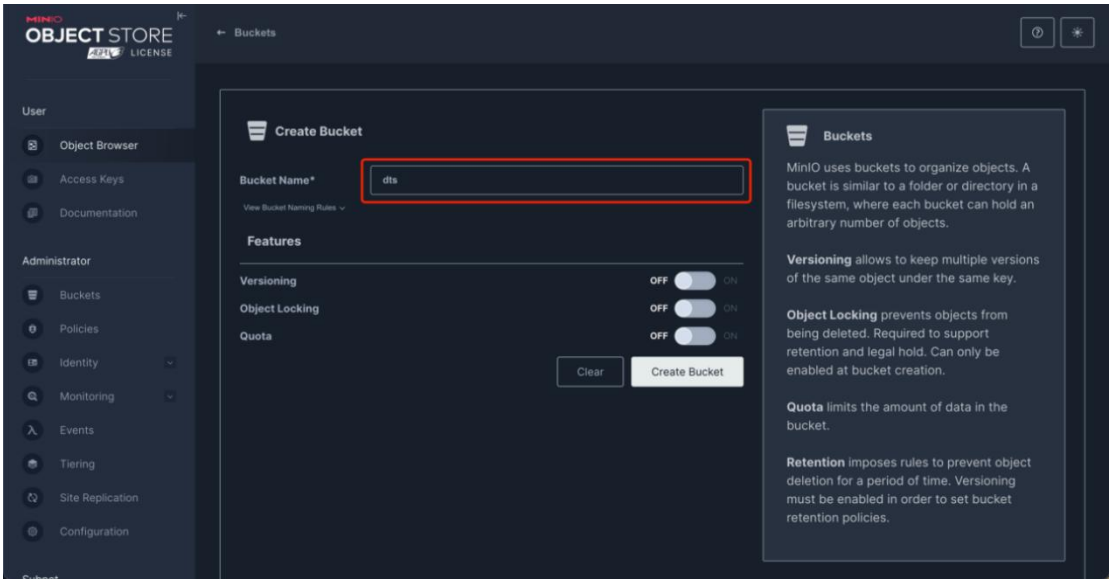
1. 使用浏览器创建密钥

1）访问 http://localhost:9001/login 用户 admin/admin123 创建 bucket、accessKey、secretKey。

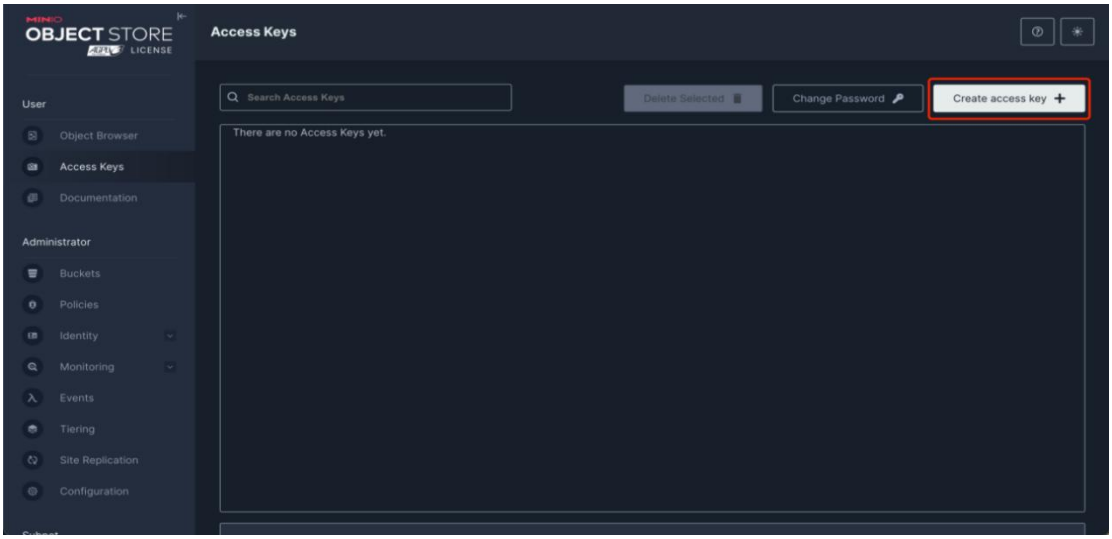


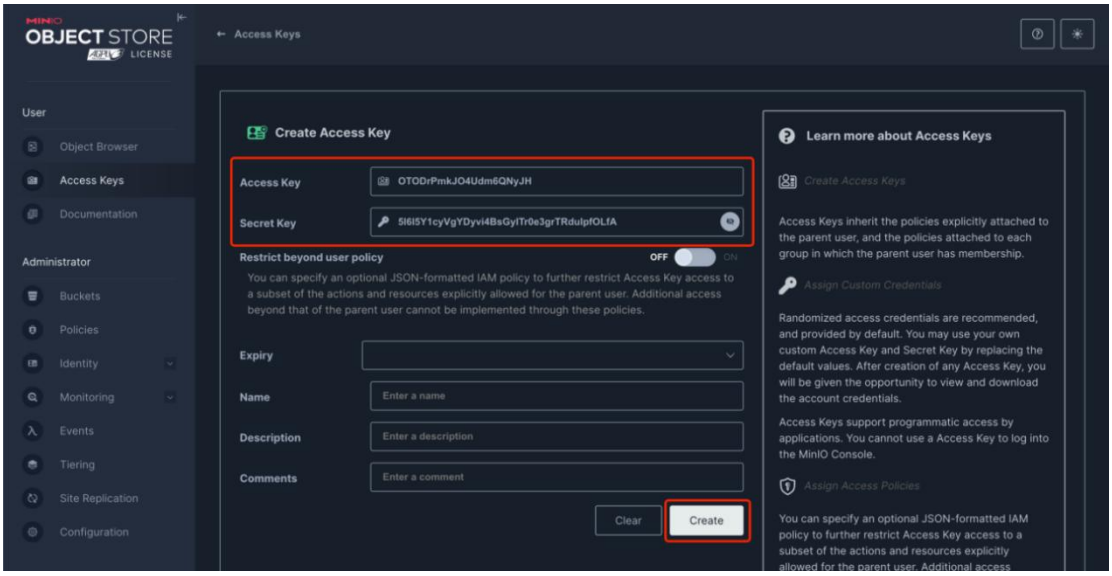
2）创建 bucket



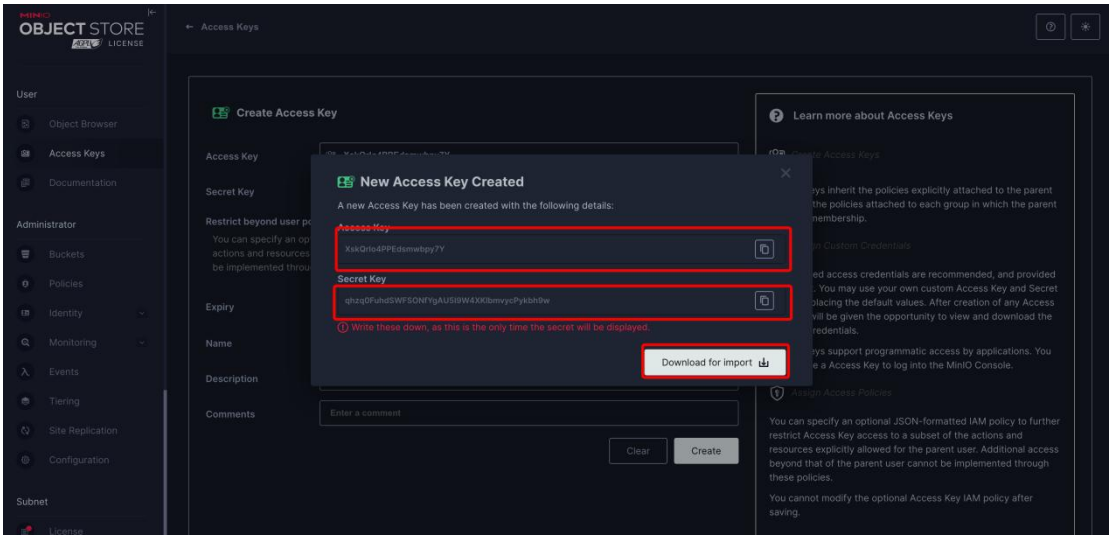


3）创建 Assess Key 、SecretKey，并记录





4）创建并保存生成的 Accesskey 和 SecretKey（或点击 Download 保存），安装 DTS 时将该密钥填入 master 配置文件中



2. 使用 mc 客户端创建密钥

通过官方下载客户端软件 “mc” 这里放在了 minio 的目录下，修改可执行权限，并链接已经安装的 minio，创建 ak、sk 和存储桶

```
Bash
chmod +x ./mc
./mc alias set dtsminio http://127.0.0.1:9000 admin admin1234
./mc admin accesskey create  dtsminio/ admin --access-key dtsaccesskey --secret-key dtssecretkey
./mc mb dtsminio/dts
```

minio 及高可用部署（可选）

参考官方手册

安装 Redis

本章节会演示使用 docker 镜像方式安装 redis 单机、使用二进制安装 redis 并配置高可用，如需配置 redis 的高可用可以通过查看的官方手册完成。镜像包可以通过官方下载可以通过万里技术支持团队获取

使用 docker 镜像方式安装 redis 单机

1. **解压**

```
Bash
tar xf redis_7.0.12.img.tar.gz
```

2. **导入镜像包**

```
toml
docker load -i redis_7.0.12.img
```

3. **查看导入的镜像**

```
toml
docker images
```

4. **配置启动 docker 镜像**

标红色部分请根据具体配置自行修改，如果启动失败，需要加上参数：--sysctl net.core.somaxconn=511 --privileged=true

```
toml
docker run --restart=always -d \
--privileged=true  -p 6379:6379  \
-v "/greatdts/redis/data:/data"  \
--name "redis" "redis:7.0.12"
```

使用二进制包方式安装 redis 及高可用部署

1. 获取二进制包

获取二进制包的方式有两种

1）通过官方网站下载

https://redis.io/downloads/

2）使用 GreatDTS 中自带的安装包解压并编译

```
Plain Text
tar zxf redis-7.0.15.tar.gz
cd redis-7.0.15
# 编译依赖 gcc
```

```
yum install -y gcc
# make MALLOC=libc
# make distclean # 编译出现错误，先执行 distclean
make
# To install these binaries in /usr/local/bin
# make PREFIX=/opt/redis install
make install
```

2. 从安装包中复制 redis.conf 到/etc 目录下，并调整如下配置

```
Shell
cp redis/conf /etc/redis.conf
vi /etc/redis.conf
# 如下配置请自行调整
protected-mode no
bind 0.0.0.0
port 6379
# 存储路径
dir /data/redis/
daemonize yes
dbfilename dump.rdb
logfile "reids.log"
```

3. 使用 systemd 来管理 redis（可选）

```
Plain Text
redis-server /etc/redis.conf
yum install systemd-devel -y
make USE_SYSTEMD=yes install
```

4. 修改配置文件/etc/redis.conf

```
Plain Text
daemonize no
supervised systemd
```

5. 创建 systemd 启动脚本

```
Plain Text
vi /usr/lib/systemd/system/redis.service
# example systemd service unit file for redis-server
```

```
#
# In order to use this as a template for providing a redis service in
your
# environment, _at the very least_ make sure to adapt the redis
configuration
# file you intend to use as needed (make sure to set "supervised
systemd"), and
# to set sane TimeoutStartSec and TimeoutStopSec property values in the
unit's
# "[Service]" section to fit your needs.
#
# Some properties, such as User= and Group=, are highly desirable for
virtually
# all deployments of redis, but cannot be provided in a manner that fits
all
# expectable environments. Some of these properties have been commented
out in
# this example service unit file, but you are highly encouraged to set
them to
# fit your needs.
#
# Please refer to systemd.unit(5), systemd.service(5), and
systemd.exec(5) for
# more information.

[Unit]
Description=Redis data structure server
Documentation=https://redis.io/documentation
#Before=your_application.service another_example_application.service
#AssertPathExists=/var/lib/redis
Wants=network-online.target
After=network-online.target

[Service]
## Alternatively, have redis-server load a configuration file:
ExecStart=/usr/local/bin/redis-server /etc/redis.conf
LimitNOFILE=10032
NoNewPrivileges=yes
#OOMScoreAdjust=-900
#PrivateTmp=yes
```

```
Type=notify
TimeoutStartSec=infinity
TimeoutStopSec=infinity
UMask=0077
#User=redis
#Group=redis
#WorkingDirectory=/var/lib/redis
```

6. **启动 redis**

```
Shell
systemctl daemon-reload
systemctl start redis
```

配置 redis 高可用（可选）

redis 存在多种高可用的集群方式，以下仅说明 sentinel 搭建高可用主备的方式（一主两从）

1）编译方式同 2.1.2.1.2 相同，这里不再重复

2）启动 master （192.168.227.11）节点，节点配置（redis.conf）如下

```
Plain Text
vi /etc/redis.conf
# 如下配置请自行调整
protected-mode
bind 0.0.0.0
port 6379
# 存储路径
dir /data/redis/
daemonize yes
dbfilename dump.rdb
logfile "reids.log"

# 如果配置了 requirepass， 则主从必须制定 masterauth
# mastarauth root123

requirepass XU21LlTCVYFXjPF9gC3YmVrOHwWjOXyr
```

3）启动 slave1（192.168.227.13）、和 slave2（192.168.227.15）

两个节点都可以使用如下配置，关键的内容是 replicaof 中的 master 地址

```
Bash
vi /etc/redis.conf
# 如下配置请自行调整
protected-mode no
bind 0.0.0.0
port 6379
# 存储路径
dir /data/redis/
daemonize yes
dbfilename dump.rdb
logfile "reids.log"

# master 节点的地址和端口
replicaof 192.168.227.11 6379
replica-read-only yes

requirepass XU21LlTCVYFXjPF9gC3YmVrOHwWjOXyr
```

4）使用 redis-server /etc/redis.conf 启动后，使用 redis-cli 连接 master，执行 info 。 有如下打印信息，说明主从复制搭建成功。

```
Bash
[root@lancer redis-7.0.15]# redis-cli
# Replication
role:master
connected_slaves:2
slave0:ip=192.168.227.15,port=6379,state=online,offset=1008,lag=0
slave1:ip=192.168.227.13,port=6379,state=online,offset=1008,lag=1
```

5）配置的 sentinel （ redis-sentinel.conf)， 三台机器 master 、slave1、slave2 都需要启动

```
Plain Text
vi /etc/redis-sentinel.conf
protected-mode no
port 26379
daemonize yes
pidfile /var/run/redis-sentinel.pid
logfile "/data/redis/sentinel.log"
```

```
dir /tmp
sentinel monitor mymaster 192.168.227.11 6379 2
sentinel down-after-milliseconds mymaster 30000
acllog-max-len 128
sentinel parallel-syncs mymaster 1
sentinel failover-timeout mymaster 30000
#sentinel deny-scripts-reconfig yes
#SENTINEL resolve-hostnames no
#SENTINEL announce-hostnames no
#SENTINEL master-reboot-down-after-period mymaster 0
```

```
Plain Text
#start sentinel
redis-sentinel /etc/redis-sentinel.conf
# 在 sentinel.log（/data/redis/sentinel.log）中相应的连接信息输出
```

3.6.5 安装 GreatDTS

本章节会演示配置一个 master 和一个 worker 。请通过商务渠道获取官方正版安装包

1. **解压 GreatDTS**

```
Bash
tar xf GreatDTS-6.1.1-GA-2-1f699add.tar.gz
```

2. **解压 master**

```
Bash
tar xf greatdts-master-6.1.1-1f699add.tar.gz
```

3. **配置 master 的配置文件，标红部分请根据实际配置修改，**

```
bash
cd master/config
vi application.properties
#
# =======元数据库 配置
#url - 一般只需要修改 主机 和 端口号
dts.mysql.url=jdbc:mysql://127.0.0.1:3310/greatdts?useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai
```

```
#账号
dts.mysql.user=root
#密码
dts.mysql.password=root1234
#连接池最大连接数
dts.mysql.max-pool-size=50
# =======minio 配置
dts.minio.endpoint=http://127.0.0.1:9000
dts.minio.bucket=dts
dts.minio.accessKey=dtsaccesskey
dts.minio.secretKey=dtssecretkey
#
# =======redis 配置 （内嵌 redis 和独立 redis 需要二选一）
#使用内嵌 redis 的开关（开：true，关：false；关后需要配置下面的独立
redis）
dts.redis.embedded.enable=false
#使用内嵌 redis 的端口号 - 根据需要改
dts.redis.embedded.port=16379
#使用内嵌 redis 的密码 - 一般不用改
dts.redis.embedded.password=XU21L1TCVYFXjPF9gC3YmVrOHwWjOXyr
#使用内嵌 redis 的数据存放目录 - 一般不用改
dts.redis.embedded.dir=${GREATDTS_MASTER_WORKSPACE}/data/
#使用内嵌 redis 的数据文件名 - 一般不用改
dts.redis.embedded.dbfilename=dump.rdb
#----
#独立 redis 的 host
dts.redis.host=127.0.0.1
#独立 redis 的端口号
dts.redis.port=6379
#独立 redis 的用户名（如果没有请注释）
#dts.redis.username=greatdts
#独立 redis 的密码（如果没有请注释）
#dts.redis.password=KJSFhyomcdgJdfdfjkfgh8
#
# ======master 配置
#web 服务端口
dts.master.web.port=7700
###rpc 相关
dts.master.rpc.address=127.0.0.1
dts.master.rpc.akka.port=10086
```

```
dts.master.rpc.http.port=10010
#----
###master jvm 参数
#最大堆空间大小（-Xmx） - 运行时所需内存超过该值会报 OOM（内存溢出），根据需要填写，如：2048m、3g、6g，4g-8g 差不多就可以了
dts.master.memory=4g
#其他 jvm 参数，支持环境变量：GREATDTS_MASTER_WORKSPACE（master 目录）、LOG_DIR（日志目录）、APPLICATION_CONF_FILE、LOGBACK_CONFIGURATION_FILE
dts.master.jvm.opts=-server -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=$LOG_DIR -XX:-UseAdaptiveSizePolicy -XX:MaxTenuringThreshold=15 -XX:+DisableExplicitGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -Djava.awt.headless=true -Djava.net.preferIPv4Stack=true -Dfile.encoding=UTF-8 -Dloader.path=$GREATDTS_MASTER_WORKSPACE/lib/ -Dlog.dir=$LOG_DIR
```

4. **启动 master**

```
Bash
cd master/bin
./startup.sh
#检查端口是否存在
ss -tnl | grep 7700
#查看启动日志
cd master/logs
tail -f greatdts-server-application.log
```

5. **配置 master 高可用（可选）**

多个 master 的配置文件中配置相同的元数据库、minio、redis 地址

6. **解压 worker**

```
Bash
tar xf greatdts-worker-6.1.1-1f699add.tar.gz
```

7. **配置 worker 的配置文件，标红部分请根据实际配置修改**

dts.worker.memory 为运行时所需内存，可根据机器配置和任务数据量调整

```
bash
cd worker/config
vi application.properties
```

```
dts.worker.ip=127.0.0.1
# Transport port, default is 27777
dts.worker.port=27777
# Address of Server node(s). Ip:port or domain. Multiple addresses should be separated with comma.
dts.worker.server-address=127.0.0.1:7700

#最大堆空间大小（-Xmx） - 运行时所需内存超过该值会报 OOM（内存溢出），根据需要填写，如：4096m、5g、10g，一般能大尽量大
dts.worker.memory=8g
#其他 jvm 参数，支持环境变量：GREATDTS_WORKER_WORKSPACE（worker 目录）、LOG_DIR（日志目录）、APPLICATION_CONF_FILE、LOGBACK_CONFIGURATION_FILE
dts.worker.jvm.opts=-server -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=$LOG_DIR -XX:SurvivorRatio=8 -XX:-UseAdaptiveSizePolicy -XX:MaxTenuringThreshold=15 -XX:+DisableExplicitGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -Djava.awt.headless=true -Djava.net.preferIPv4Stack=true -Dfile.encoding=UTF-8 -Dloader.path=$GREATDTS_WORKER_WORKSPACE/lib/ -Dlog.dir=$LOG_DIR
#仅调试&查看 GC 时开启（开启或关闭后需要重启 worker 才能生效）
#dts.worker.jvm.gc=-XX:+PrintGCDateStamps -XX:+PrintTenuringDistribution -XX:+PrintGCApplicationStoppedTime -XX:+PrintGCApplicationConcurrentTime -XX:+PrintGCDetails -Xloggc:$LOG_DIR/gc.log
```

8. **启动 worker**

```
Bash
cd worker/bin
./startup.sh
#检查端口是否存在
ss -tnl | grep 27777
#查看日志
tail -f logs/greatdts-worker-all.log
```

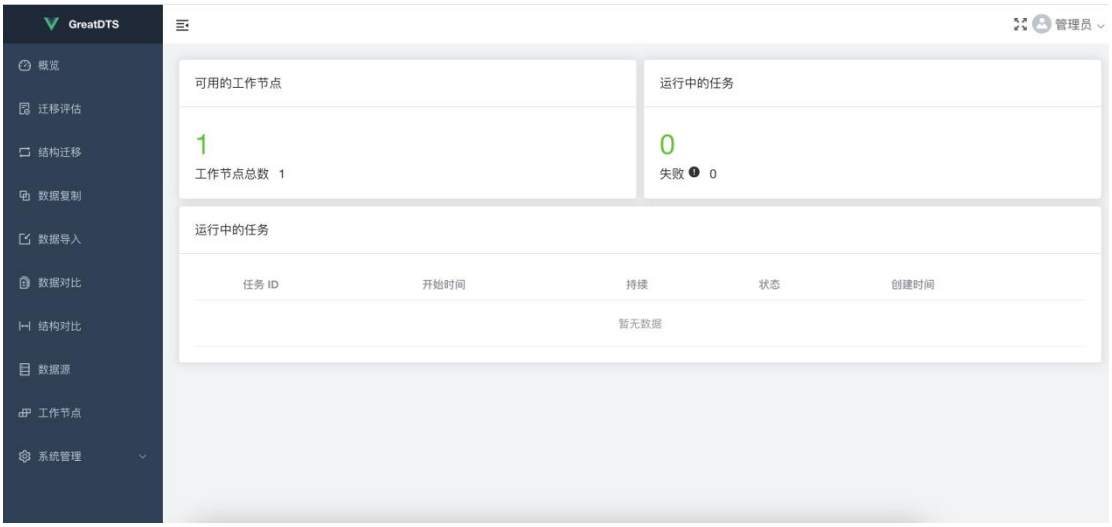
9. **配置 worker 高可用（可选）**

如下所示，在 worker 配置文件中配置多个 master 地址注意是双引号中多个 master 使用逗号分隔

```
Plain Text
dts.worker.server-address="master1:7700,master2:7700,master3:7700"
```

3.6.6 登录 GreatDTS

1. 登录地址：http://本地 IP 地址:7700
2. 默认用户名密码：admin/123456



4. 核心功能

4.1 用户管理

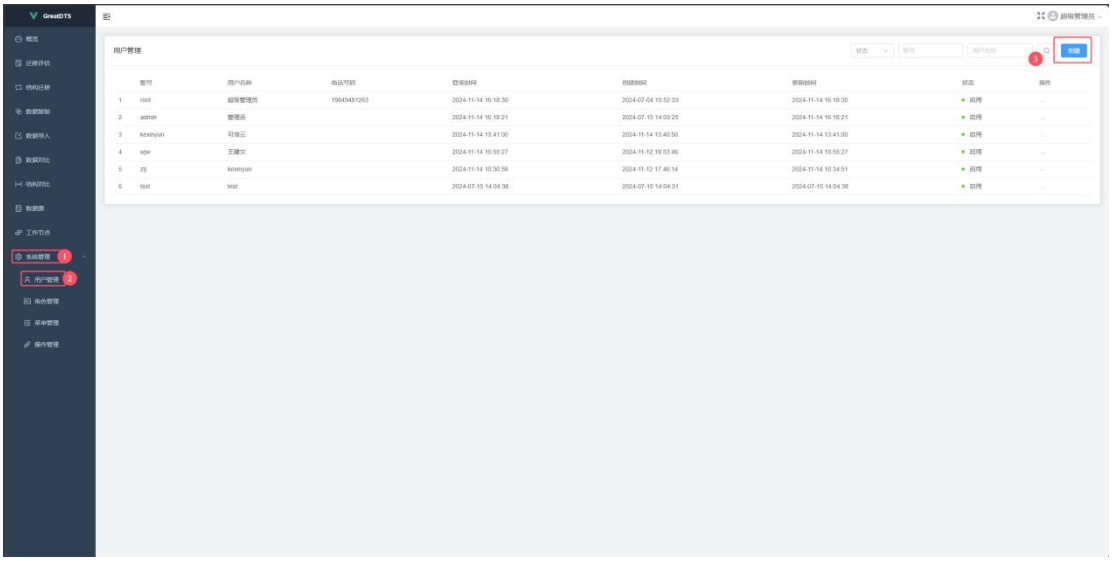
1. 用户体系介绍

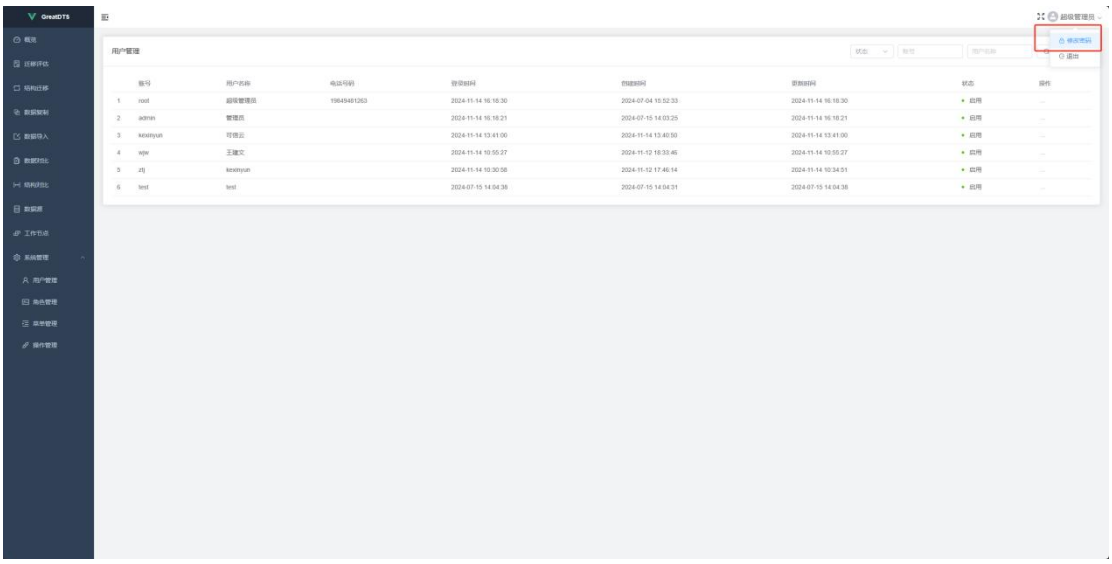
为方便用户初始化 GreatDTS 内置三类角色和两个管理员，三类角色为超级管理员、管理员、普通用户。两个内置管理员为超级管理员 root 和管理员 admin，两个默认密码都为 123456，首次登录后需自行修改。如下为用户体系的详细介绍

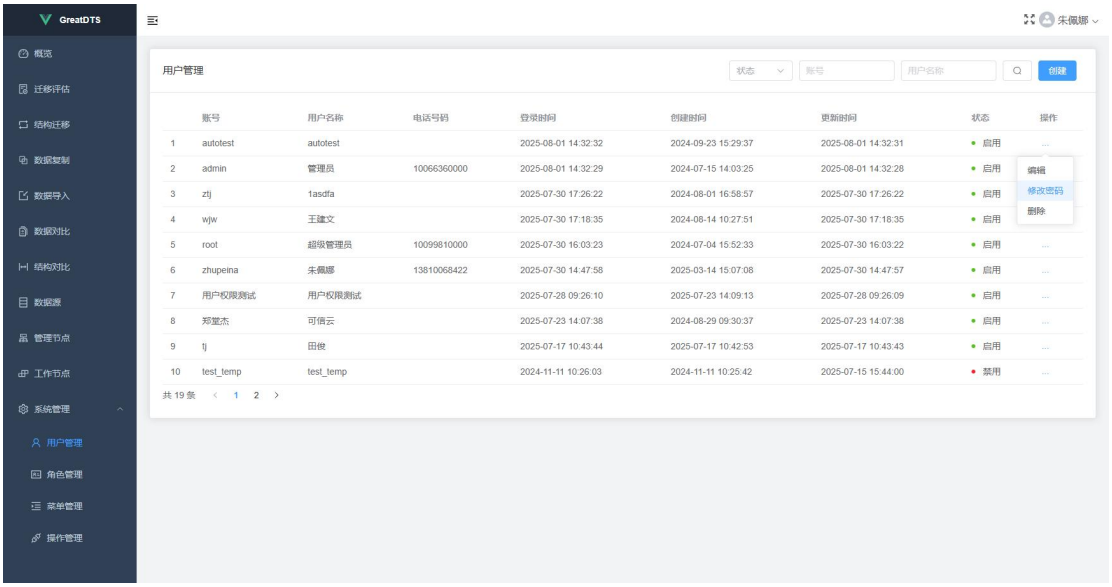
账号	root	admin	用户自定义
昵称	超级管理员	管理员	用户自定义
角色	超级管理员	管理员	普通用户
是否内置	是	是	否
是否唯一	是，有且只有一个超级管理员	否，可以被创建	否可以被创建
修改密码	自身、管理员、普通用户	自身、其他管理员、普通用户	自身

角色变更	管理员、普通用户	其他管理员、普通用户	不支持
手机号变更	自身、管理员、普通用户	自身、其他管理员、普通用户	不支持
邮箱变更	自身、管理员、普通用户	自身、其他管理员、普通用户	不支持
启禁用	管理员、普通用户	自身、管理员、普通用户	不支持
菜单权限	有	有	无系统管理菜单

2. 用户管理功能展示







密码复杂度校验： 必须包含大小写字母数字特殊符号长度为 8-16 位

4.2 数据源管理

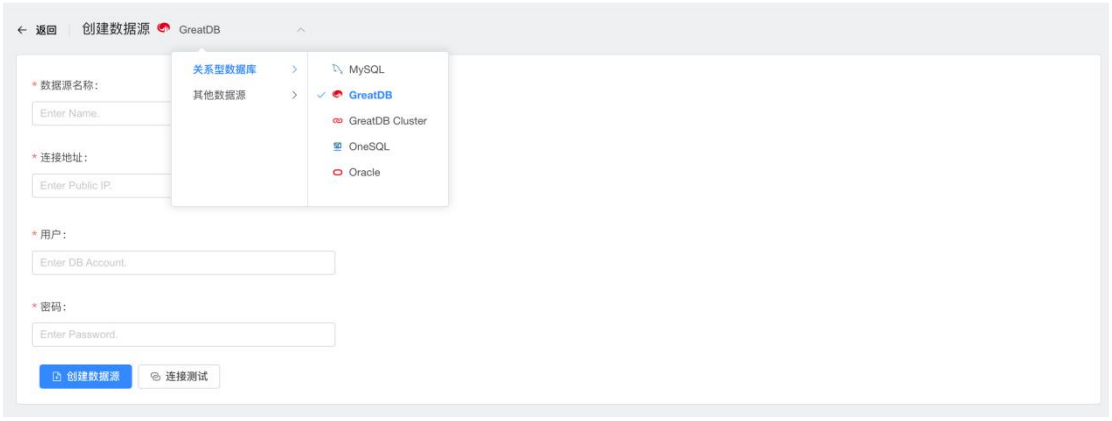
GreatDTS 支持将不同类型的数据源添加到控制台中进行统一管理，您可以对已经添加后的数据源使用迁移评估、结构迁移、数据复制、数据库对比等功能。

4.2.1 创建数据源

- 当前支持的数据源类型：GreatDB、GreatDB Cluster 、MySQL、Oracle、OneSQL、Kafka、ClickHouse

注：数据源创建后，数据源的类型不可修改

如下为操作示例



- 数据要素

参数	说明
数据源名称	数据源名称
连接地址	数据源的 IP 地址和端口
连接方式	当数据源类型为 Oracle 时，需要选择连接方式 SID 或 Service Name
用户	数据源的用户名，GreatDTS 将使用此用户完成迁移评估、结构迁移、数据复制、结构比对、数据比对的各项任务
密码	数据源的密码

4.2.2 最小化权限

GreatDTS 将使用创建数据源时配置的用户完成迁移评估、结构迁移、数据复制、结构比对、数据比对的各项任务，如下是不同数据源类型，完成不同任务需要的最小化权限。为了进一步保障数据库安全，根据使用的不同功能精细化赋权。

数据源	类型	迁移评估	结构迁移	全量复制	增量复制	结构对比	数据对比
MySQL 5.7+	源	GRANT SHOW DATABASES, SELECT ON *.* TO 'user_dts@'%';	GRANT SHOW DATABASES, SELECT, SHOW VIEW, TRIGGER, EVENT ON *.* TO 'user_dts@'%';	GRANT SELECT,REPLICATION CLIENT ON *.* TO 'user_dts'@'%';	GRANT SELECT, REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'user_dts'@'%';	GRANT SHOW DATABASES,SELECT ON *.* TO 'user_	GRANT SHOW DATABASES,SELECT ON *.* TO 'user_

						dts@'%';	dts@'%';
	目标	GRANT SHOW DATABASES, SELECT ON *.* TO 'user_dts'@'%';	GRANT CREATE,DROP,CREATE VIEW,CREATE ROUTINE,TRIGGER,EVENT,SUPER, SELECT ON *.* TO 'user_dts'@'%';	GRANT DROP, SELECT,INSERT,UPDATE,DELETE ON *.* TO 'user_dts'@'%';	GRANT CREATE,DROP,ALTER SELECT,INSERT,UPDATE,DELETE ON *.* TO 'user_dts'@'%'; ON *.* TO 'user_dts'@'%';	GRANT SHOW DATABASES,SELECT ON *.* TO 'user_dts@'%';	GRANT SHOW DATABASES,SELECT ON *.* TO 'user_dts@'%';
Oracle 11G+	源	GRANT CREATE SESSION TO user_dts; GRANT EXECUTE ON DBMS_METADATA TO user_dts; GRANT SELECT ANY DICTIONARY TO	GRANT CREATE SESSION TO user_dts; GRANT EXECUTE ON DBMS_METADATA TO user_dts; GRANT SELECT ANY	#oracle11g GRANT CREATE SESSION TO <user_dts>; GRANT SELECT ON V_$DATABASE to <user_dts>; ≦ GRANT FLASHBACK ANY TABLE TO	开启归档、开启附加日志 #oracle11g GRANT CREATE SESSION TO <user_dts>; GRANT SELECT ON V_$DATABASE to	GRANT CREATE SESSION TO user_dts; GRANT EXE	GRANT CREATE SESSION TO user_dts; GRANT EXE

		user_dts; GRANT SELECT_CATALOG_ROLE TO user_dts; GRANT SELECT ON V_$LOG TO <user_dts> ; GRANT SELECT ON V_$LOG_HISTORY TO <user_dts> ; GRANT SELECT ON V_$LOGMNR_LOGS TO <user_dts> ; GRANT SELECT ON V_$LOGMNR_CONTENTS TO <user_dts> ; GRANT SELECT ON V_$LOGMNR_PARAMETERS TO <user_dts> ; GRANT	DICTIONARY TO user_dts; GRANT SELECT_CATALOG_ROLE TO user_dts; GRANT SELECT ON V_$LOG TO <user_dts> ; GRANT SELECT ON V_$LOG_HISTORY TO <user_dts> ; GRANT SELECT ON V_$LOGMNR_LOGS TO <user_dts> ; GRANT SELECT ON V_$LOGMNR_CONTENTS TO <user_dts> ; GRANT SELECT ON V_$LOGMNR_PARAMETERS TO <user_dts	<user_dts>; GRANT SELECT ANY TABLE TO <user_dts>; GRANT SELECT_CATALOG_ROLE TO <user_dts>; GRANT EXECUTE_CATALOG_ROLE TO <user_dts>; GRANT SELECT ANY TRANSACTION TO <user_dts>; GRANT LOCK ANY TABLE TO <user_dts>; GRANT SELECT ON V_$LOG TO <user_dts> ; GRANT SELECT ON V_$LOG_HISTORY TO <user_dts> ; GRANT SELECT ON V_$LOGMNR_LOGS TO <user_dts> ; GRANT SELECT ON V_$LOGMNR_CONTENTS TO	<user_dts> ; GRANT FLASHBACK ANY TABLE TO <user_dts> ; GRANT SELECT ANY TABLE TO <user_dts> ; GRANT SELECT_CATALOG_ROLE TO <user_dts> ; GRANT EXECUTE_CATALOG_ROLE TO <user_dts> ; GRANT SELECT ANY TRANSACTION TO <user_dts> ; GRANT LOGMINING TO <user_dts> ; GRANT CREATE	CUTE ON DBMS_METADATA TO user_dts; GRANT SELECT ANY DICTIONARY TO user_dts; SELECT ON V_$DATABASE TO user_dts	CUTE ON DBMS_METADATA TO user_dts; GRANT SELECT ANY DICTIONARY,SELECT ANY TABLE TO user_dts; SELECT ON V_$DATABASE TO use

		SELECT ON V_$LOGFILE TO <user_dts> ; GRANT SELECT ON V_$ARCHIVED_LOG TO <user_dts> ; GRANT SELECT ON V_$ARCHIVE_DEST_STATUS TO <user_dts> ;	> ; GRANT SELECT ON V_$LOGFILE TO <user_dts> ; GRANT SELECT ON V_$ARCHIVED_LOG TO <user_dts> ; GRANT SELECT ON V_$ARCHIVE_DEST_STATUS TO <user_dts> ;	<user_dts> ; GRANT SELECT ON V_$LOGMNR_PARAMETERS TO <user_dts> ; GRANT SELECT ON V_$LOGFILE TO <user_dts> ; GRANT SELECT ON V_$ARCHIVED_LOG TO <user_dts> ; GRANT SELECT ON V_$ARCHIVE_DEST_STATUS TO <user_dts> ; #oracle19c GRANT CREATE SESSION TO <user_dts> CONTAINER=ALL; GRANT SET CONTAINER TO <user_dts> CONTAINER=ALL; GRANT SELECT ON V_$DATABASE	TABLE TO <user_dts> ; GRANT LOCK ANY TABLE TO <user_dts> ; GRANT ALTER ANY TABLE TO <user_dts> ; GRANT CREATE SEQUENCE TO <user_dts> ; GRANT EXECUTE ON DBMS_LOGMNR TO <user_dts> ; GRANT EXECUTE ON DBMS_LOGMNR_D TO <user_dts> ; GRANT SELECT ON V_$LOG TO <user_dts> ; GRANT SELECT ON		r_dts

				to <user_dts> CONTAINER=ALL; GRANT FLASHBACK ANY TABLE TO <user_dts> CONTAINER=ALL; GRANT SELECT ANY TABLE TO <user_dts> CONTAINER=ALL; GRANT SELECT_CATALOG_ROLE TO <user_dts> CONTAINER=ALL; GRANT EXECUTE_CATALOG_ROLE TO <user_dts> CONTAINER=ALL; GRANT SELECT ANY TRANSACTION TO <user_dts> CONTAINER=ALL; GRANT LOCK ANY TABLE TO <user_dts> CONTAINER=ALL	V_$LOG_HISTORY TO <user_dts>; GRANT SELECT ON V_$LOGMNR_LOGS TO <user_dts>; GRANT SELECT ON V_$LOGMNR_CONTENTS TO <user_dts>; GRANT SELECT ON V_$LOGMNR_PARAMETERS TO <user_dts>; GRANT SELECT ON V_$LOGFILE TO <user_dts>; GRANT SELECT ON V_$ARCHIVED_LOG TO <user_dts>; GRANT SELECT ON		

				L; GRANT SELECT ON V_$LOG TO <user_dts> CONTAINER=ALL; GRANT SELECT ON V_$LOG_HISTORY TO <user_dts> CONTAINER=ALL; GRANT SELECT ON V_$LOGMNR_LOGS TO <user_dts> CONTAINER=ALL; GRANT SELECT ON V_$LOGMNR_CONTENTS TO <user_dts> CONTAINER=ALL; GRANT SELECT ON V_$LOGMNR_PARAMETERS TO <user_dts> CONTAINER=ALL; GRANT SELECT ON V_$LOGFILE TO	V_$ARCHIVE_DEST_STATUS TO <user_dts> ; #oracle19c GRANT CREATE SESSION TO <user_dts> CONTAINER=ALL; GRANT SET CONTAINER TO <user_dts> CONTAINER=ALL; GRANT SELECT ON V_$DATABASE to <user_dts> CONTAINER=ALL; GRANT FLASHBACK ANY TABLE TO <user_dts> CONTAINER=ALL; GRANT SELECT ANY TABLE TO <user_dts> CONTAINER=		

				<user_dts> CONTAINER=ALL; GRANT SELECT ON V_$ARCHIVED_LOG TO <user_dts> CONTAINER=ALL; GRANT SELECT ON V_$ARCHIVE_DEST_STATUS TO <user_dts> CONTAINER=ALL;	ALL; GRANT SELECT_CATALOG_ROLE TO <user_dts> CONTAINER=ALL; GRANT EXECUTE_CATALOG_ROLE TO <user_dts> CONTAINER=ALL; GRANT SELECT ANY TRANSACTION TO <user_dts> CONTAINER=ALL; GRANT LOGMINING TO <user_dts> CONTAINER=ALL; GRANT CREATE TABLE TO <user_dts> CONTAINER=ALL; GRANT LOCK ANY TABLE TO <user_dts>		

					CONTAINER=ALL; GRANT ALTER ANY TABLE TO <user_dts> CONTAINER=ALL; GRANT CREATE SEQUENCE TO <user_dts> CONTAINER=ALL; GRANT EXECUTE ON DBMS_LOGMNR TO <user_dts> CONTAINER=ALL; GRANT EXECUTE ON DBMS_LOGMNR_D TO <user_dts> CONTAINER=ALL; GRANT SELECT ON V_$LOG TO <user_dts> CONTAINER=ALL; GRANT SELECT ON V_$LOG_HIS		

					TORY TO <user_dts> CONTAINER= ALL; GRANT SELECT ON V_$LOGMNR_ LOGS TO <user_dts> CONTAINER= ALL; GRANT SELECT ON V_$LOGMNR_ CONTENTS TO <user_dts> CONTAINER= ALL; GRANT SELECT ON V_$LOGMNR_ PARAMETERS TO <user_dts> CONTAINER= ALL; GRANT SELECT ON V_$LOGFILE TO <user_dts> CONTAINER= ALL; GRANT SELECT ON V_$ARCHIVE D_LOG TO		

					<user_dts> CONTAINER= ALL; GRANT SELECT ON V_$ARCHIVE _DEST_STAT US TO <user_dts> CONTAINER= ALL; # 表空间的限制 ALTER USER <user_dts> QUOTA UNLIMITED on users		
	目标	不支持	不支持	GRANT CREATE SESSION, CREATE TABLE, DROP ANY TABLE, CREATE ANY VIEW, CREATE ANY PROCEDURE, CREATE ANY FUNCTION, SELECT ANY TABLE ,INSER T ANY TABLE ,UPDAT E ANY TABLE , DELETE ANY	GRANT CREATE SESSION, CREATE TABLE, DROP ANY TABLE, CREATE ANY VIEW, CREATE ANY PROCEDURE, CREATE ANY FUNCTION, CREATE ANY PROCEDURE, SELECT ANY TABLE ,INS	GRA NT CRE ATE SES SIO N TO use r_d ts; GRA NT EXE CUT E ON DBM	GRA NT CRE ATE SES SIO N TO use r_d ts; GRA NT EXE CUT E ON DBM

				TABLE TO user_dts; GRANT SELECT ON V_$DATABASE TO user_dts; GRANT SELECT ANY DICTIONARY TO user_dts;	ERT ANY TABLE ,UPDATE ANY TABLE , DELETE ANY TABLE TO user_dts; GRANT SELECT ON V_$DATABASE TO user_dts; GRANT SELECT ANY DICTIONARY TO user_dts;	S_METADATA TO user_dts; GRANT SELECT ANY DICTIONARY TO user_dts; GRANT SELECT ANY TABLE TO user_dts; GRANT SELECT ON V_$DAT	S_METADATA TO user_dts; GRANT SELECT ANY DICTIONARY,SELECT ANY TABLE TO user_dts; GRANT SELECT ANY TABLE TO user_dts; GRANT

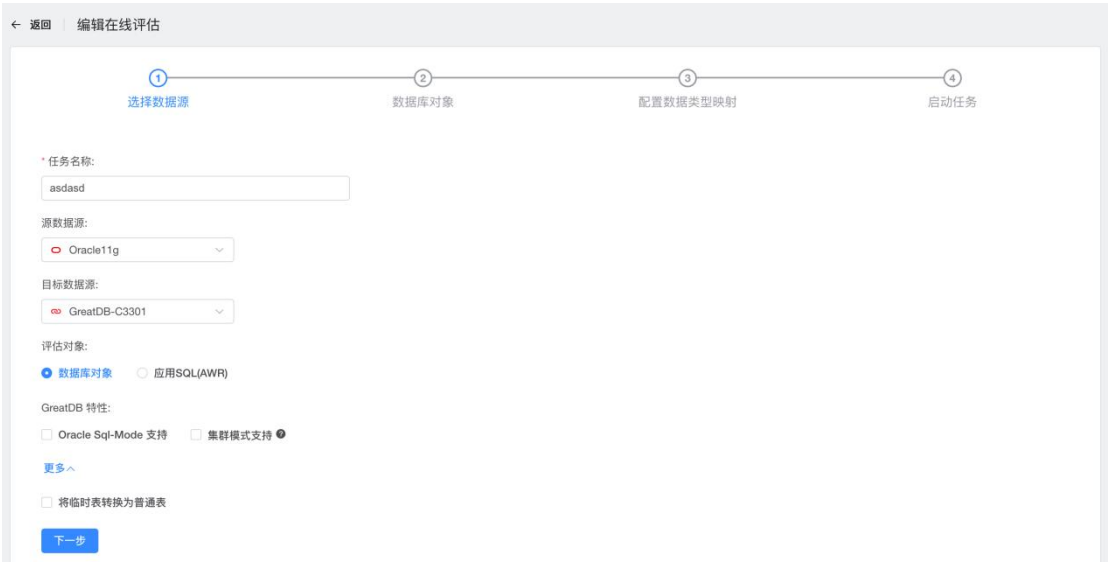
						ABASE TO user_dts	SELECT ON V_$DATABASE TO user_dts
GreatDB 5.x/6.x	源	GRANT SHOW DATABASES, SELECT ON *.* TO 'user_dts@'%';	GRANT SHOW DATABASES, SELECT, SHOW VIEW, TRIGGER, EVENT ON *.* TO 'user_dts@'%';	GRANT SELECT,REPLICATION CLIENT ON *.* TO 'user_dts'@'%';	GRANT SELECT, REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'user_dts'@'%';	GRANT SHOW DATABASES,SELECT ON *.* TO 'user_dts@'%';	GRANT SHOW DATABASES,SELECT ON *.* TO 'user_dts@'%';
	目标	GRANT SHOW DATABASES, SELECT ON *.* TO 'user_dts'@'%';	GRANT CREATE,CREATE VIEW,CREATE ROUTINE,TRIGGER,EVENT,SUPER	GRANT DROP, SELECT,INSERT,UPDATE,DELETE ON *.* TO 'user_dts'@'%';	GRANT CREATE,DROP,ALTER SELECT,INSERT,UPDATE,DELETE ON *.* TO 'user_dts'	GRANT SHOW DATABASES,SE	GRANT SHOW DATABASES,SE

			, SELECT ON *.* TO 'user_dts'@'%';		@'%'; ON *.* TO 'user_dts'@'%';	LECT ON *.* TO 'user_dts@'%';	LECT ON *.* TO 'user_dts@'%';
Click House	源						
	目标						

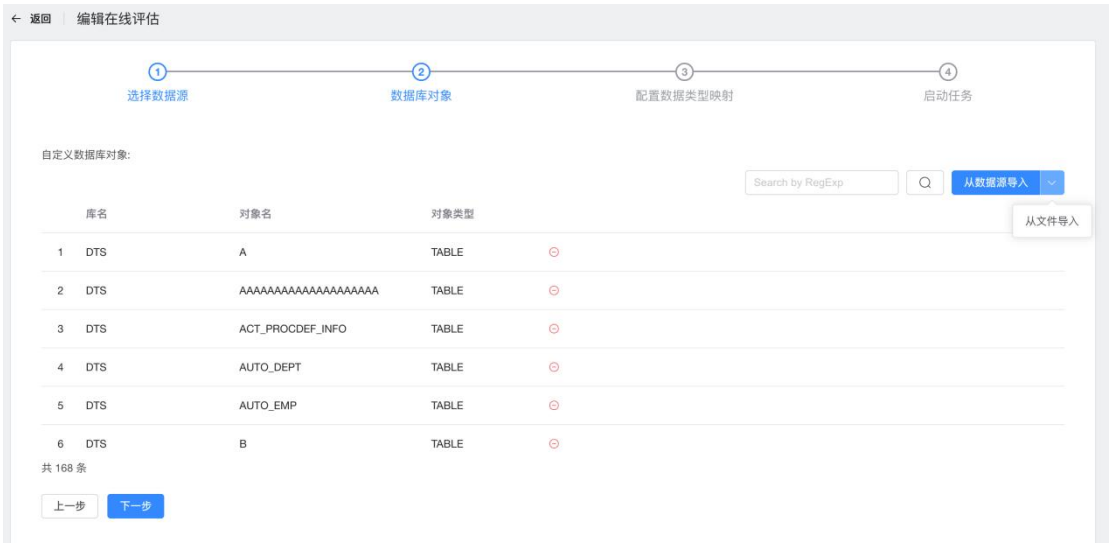
4.3 迁移评估

4.3.1 创建迁移评估

1. 选择数据源



2. 选择或导入评估对象

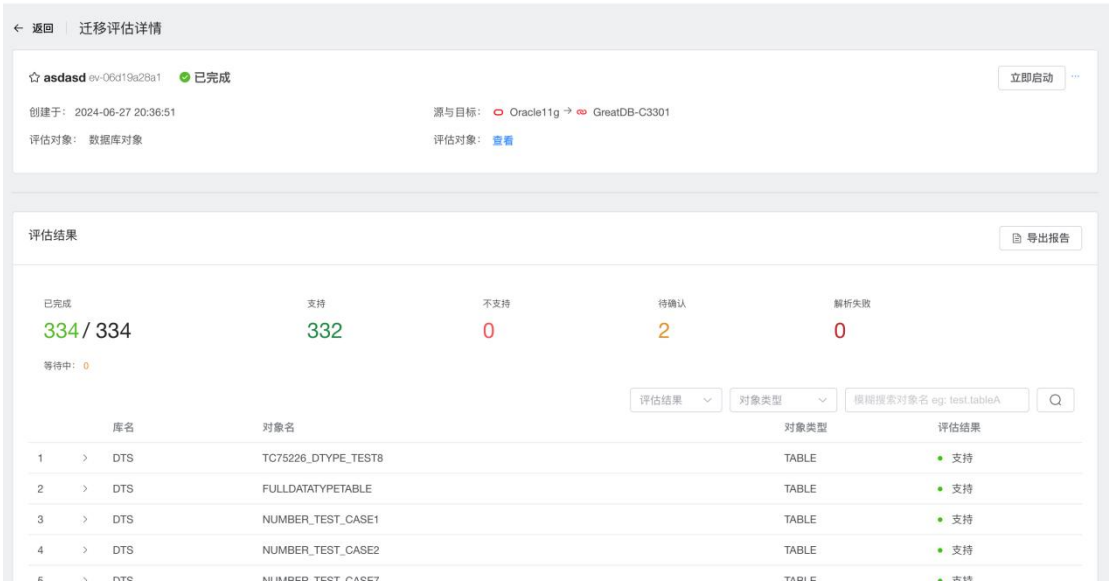


3. 配置映射

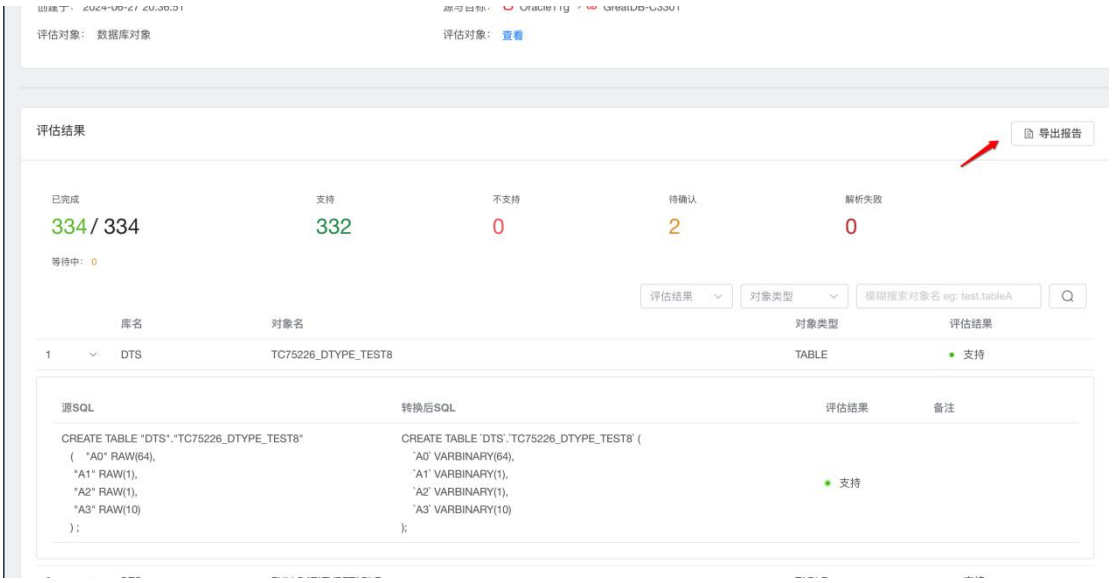
配置自定义类型映射，不配置时，程序自动使用默认的类型映射规则。

4.3.2 启动

从列表页或详情页启动评估任务



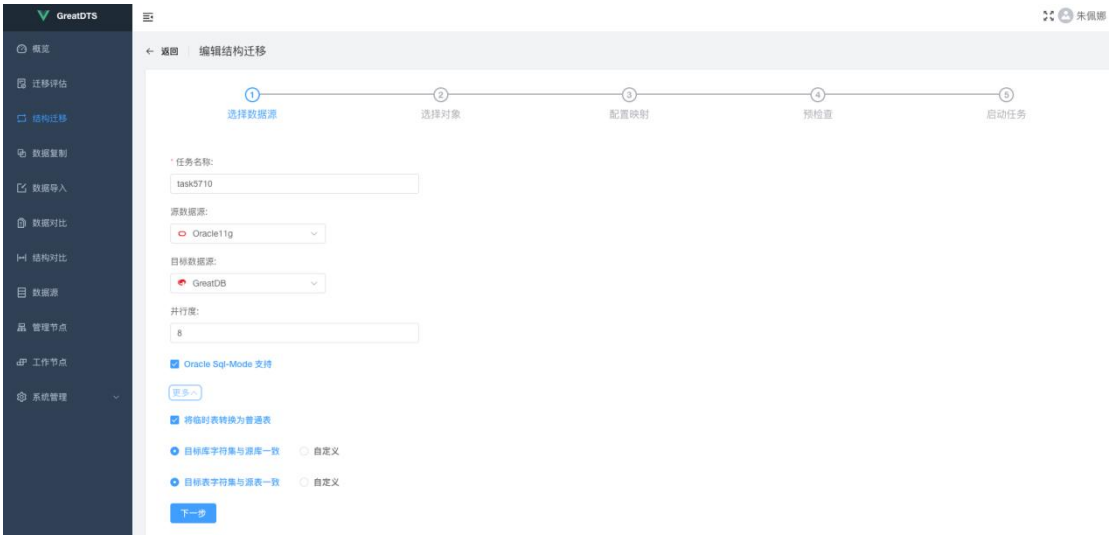
下载评估报告



4.4 结构迁移

4.4.1 创建结构迁移

1. 选择数据源

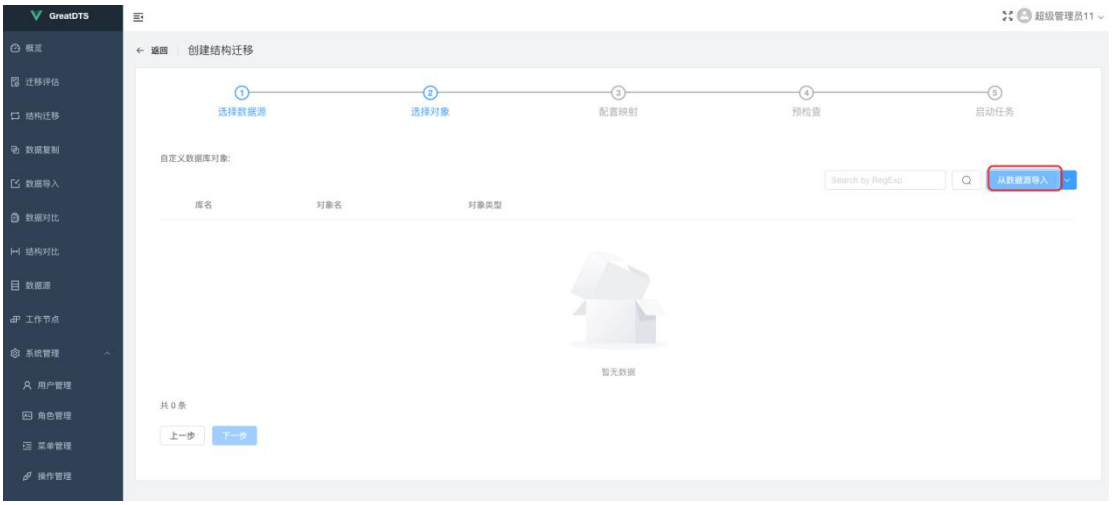


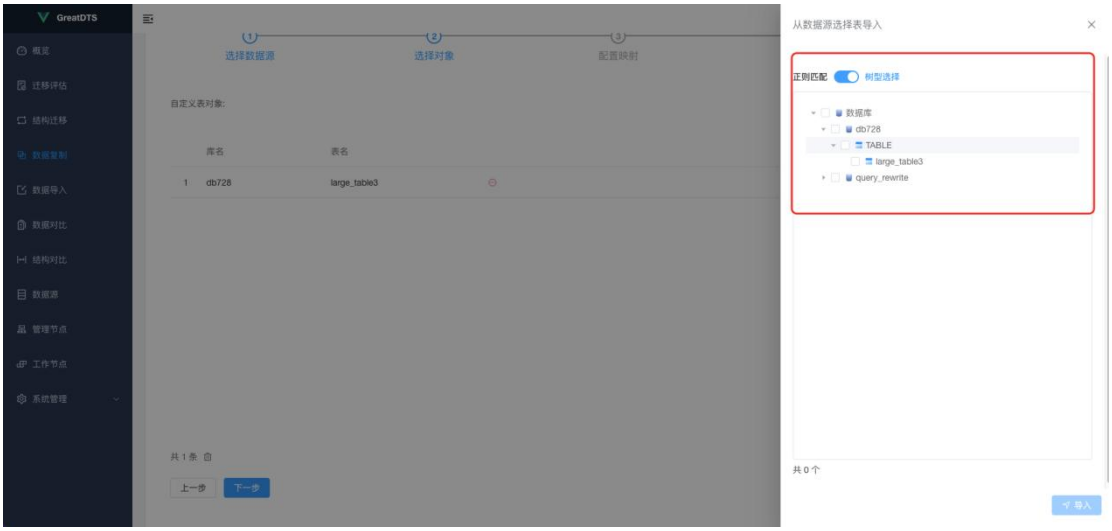
参数说明

参数	说明
并行度	并行读取转换结构数据

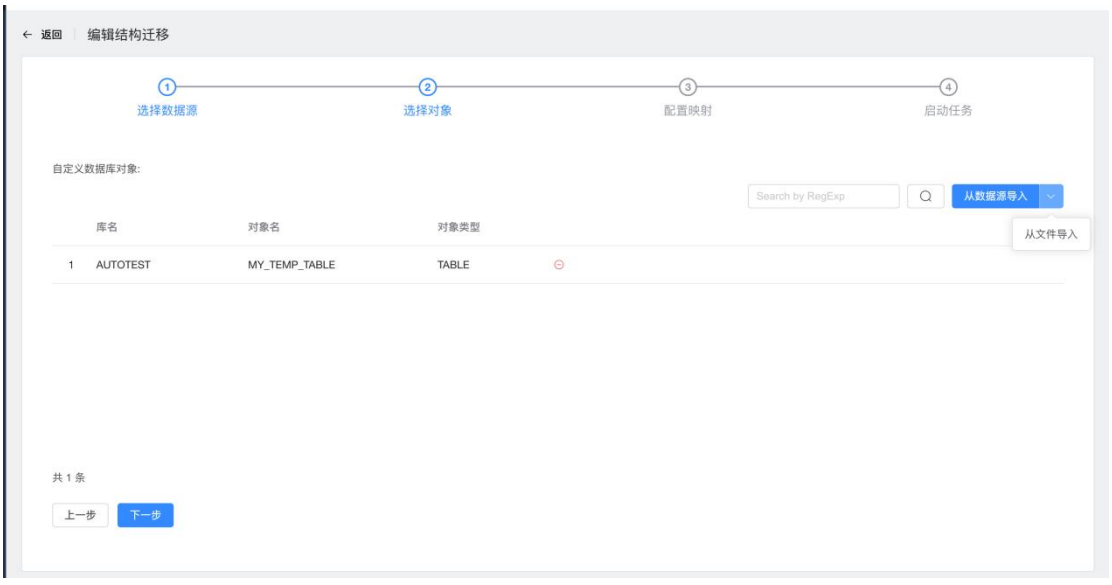
2. 选择对象

- 从数据源导入，支持在页面上直接选择对象，也支持正则表达式匹配对象。



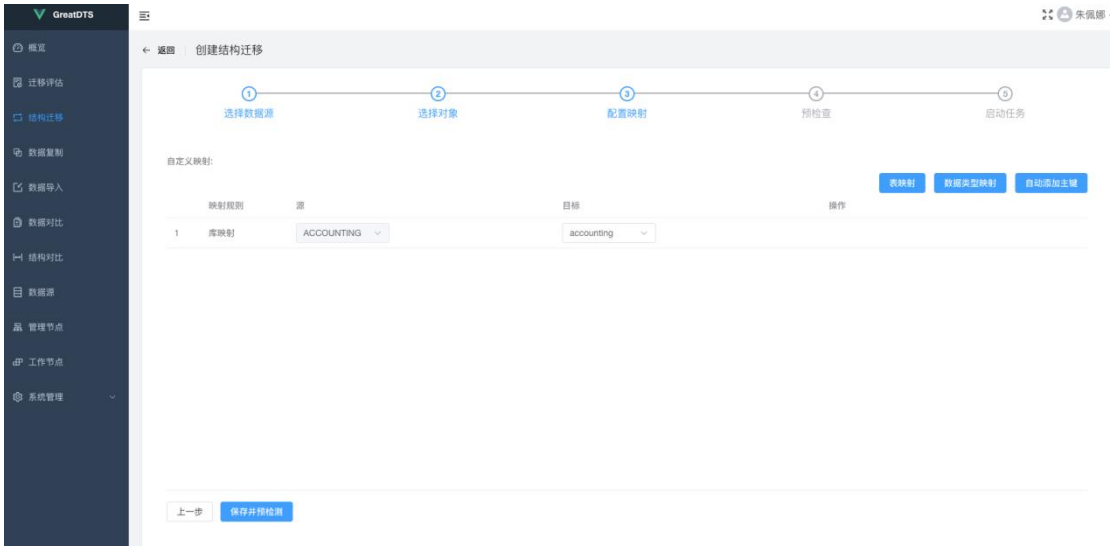


- 从文件导入，将需要导入的对象填入 execl 中点击从文件导入上传至 GreatDTS 平台

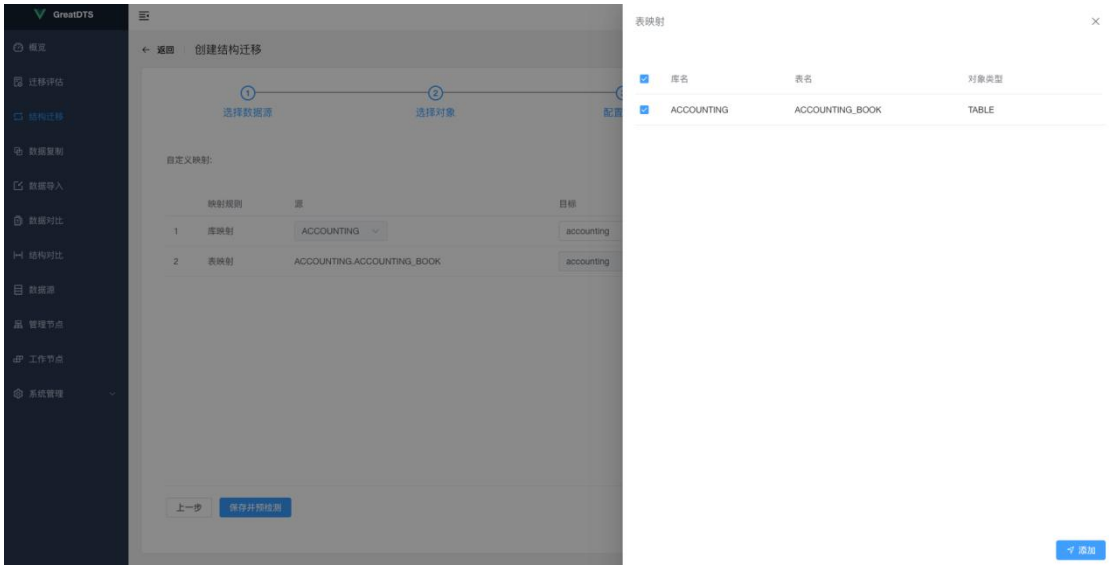


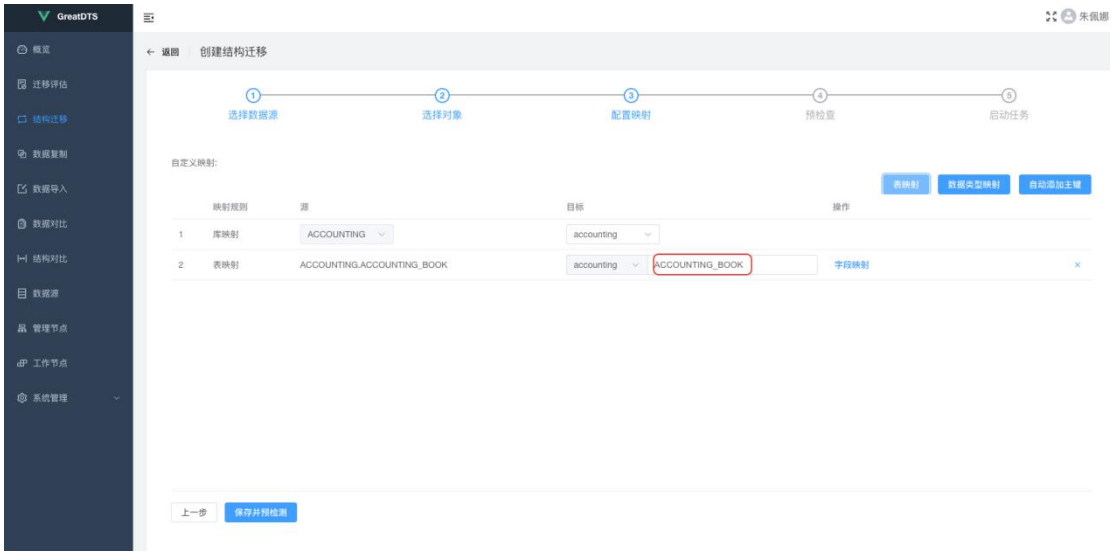
3. 配置映射

配置表映射、字段名称映射、数据类型映射、自动添加主键

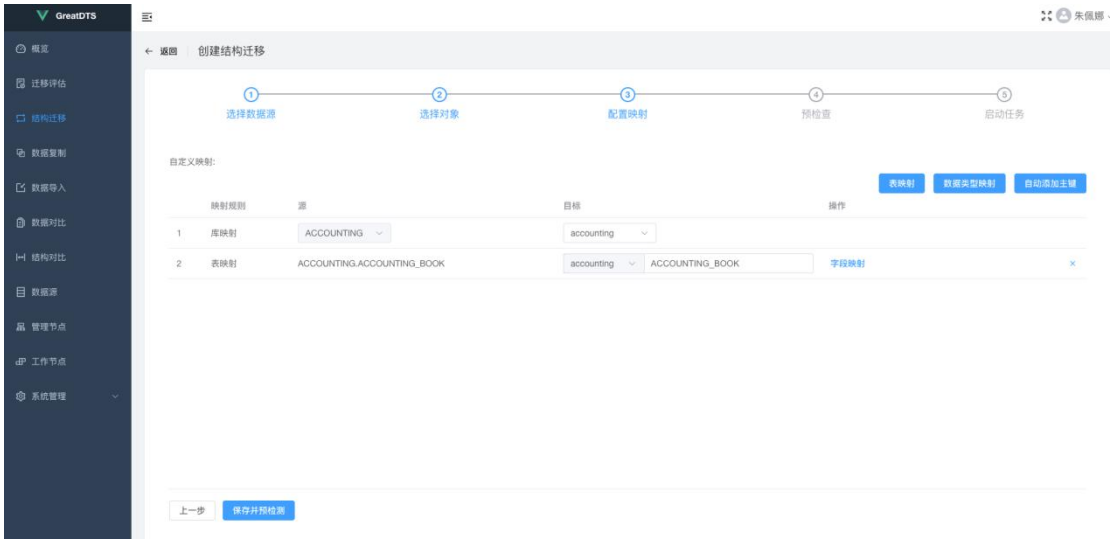


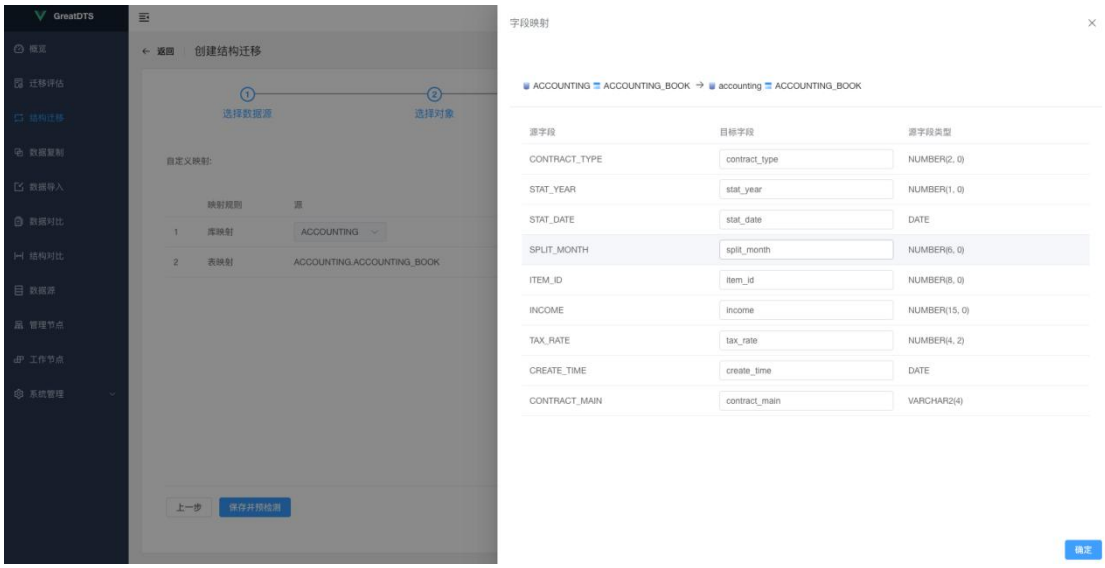
表映射：可以配置源和目标端的表名的映射关系，先点击表映射添加表，再配置表映射。如不配置表映射，默认认为源和目标端的表名一致。





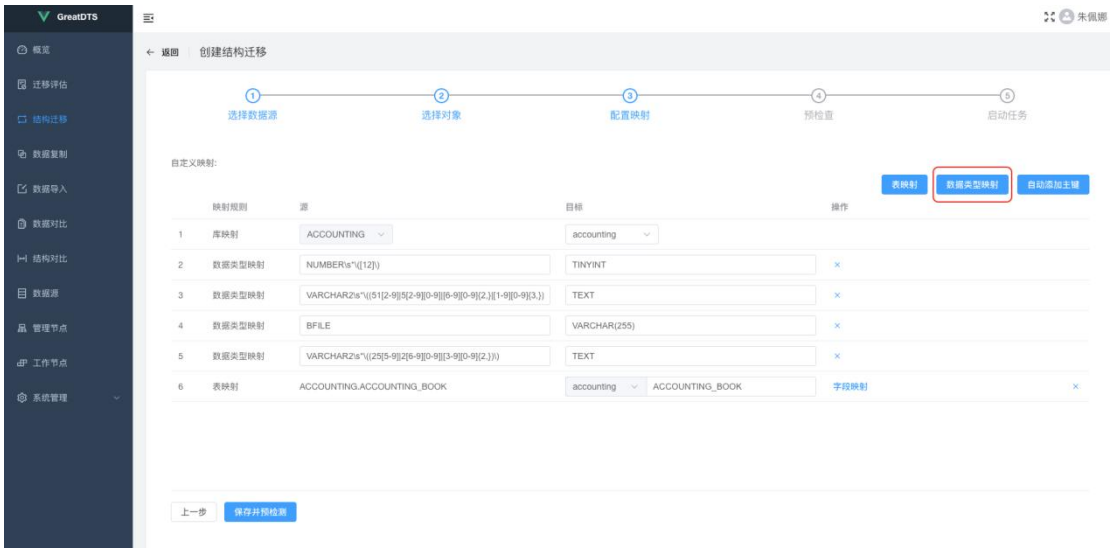
字段类型映射：添加表映射后可以配置字段映射，如不配置表映射，默认认为源和目标端的字段名称一致，不支持源和目标端字段数量不一致的场景。

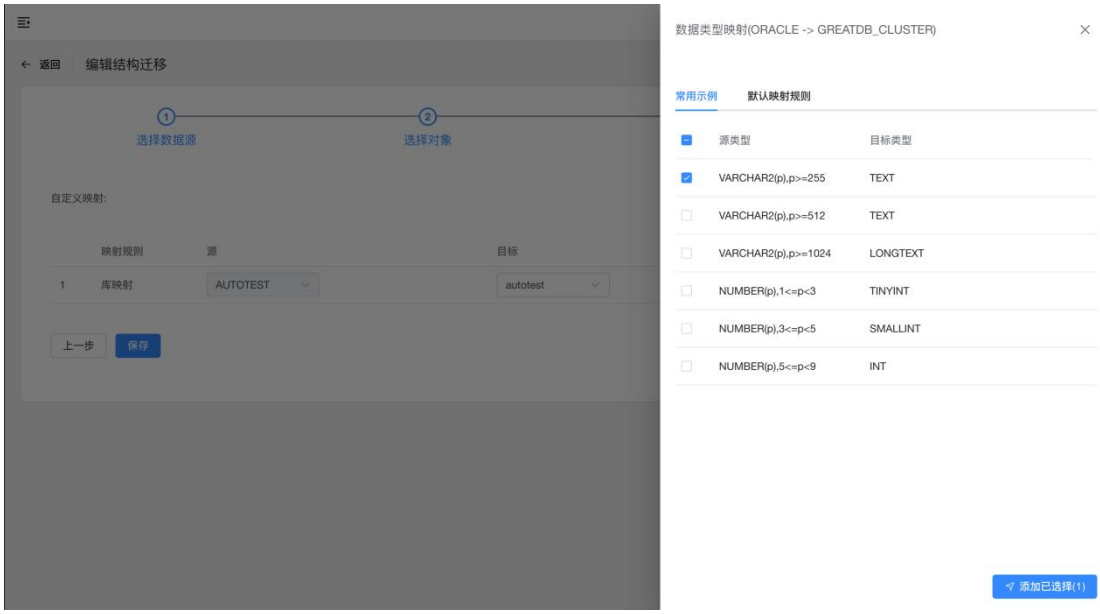


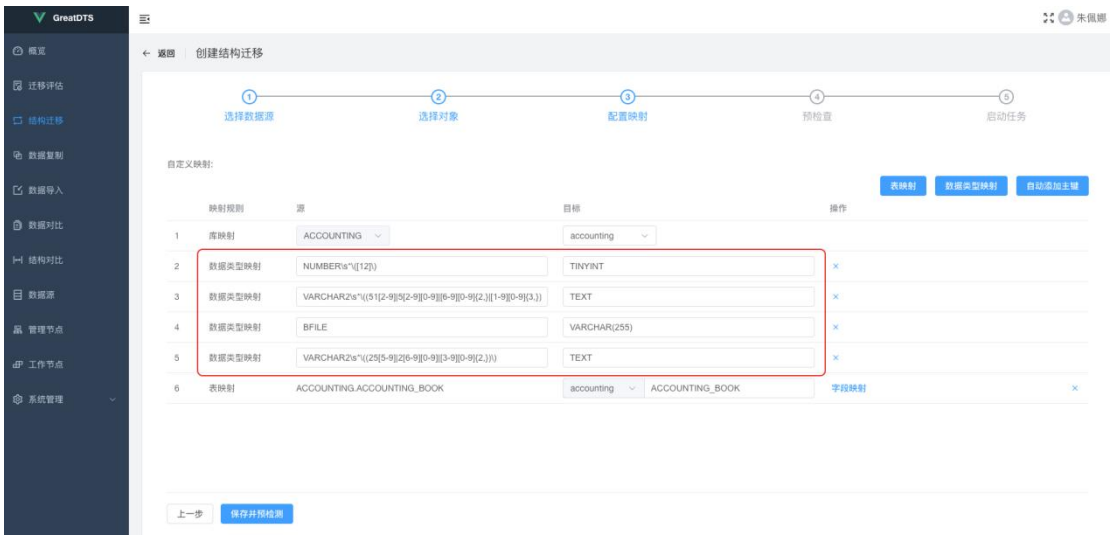


数据类型映射：点击数据映射，选择常用示例中的映射配置，可以基于原有的映

射配置做修改，目前修改仅支持正则表达式。如未配置任何数据类型映射，则使用默认映射规则。

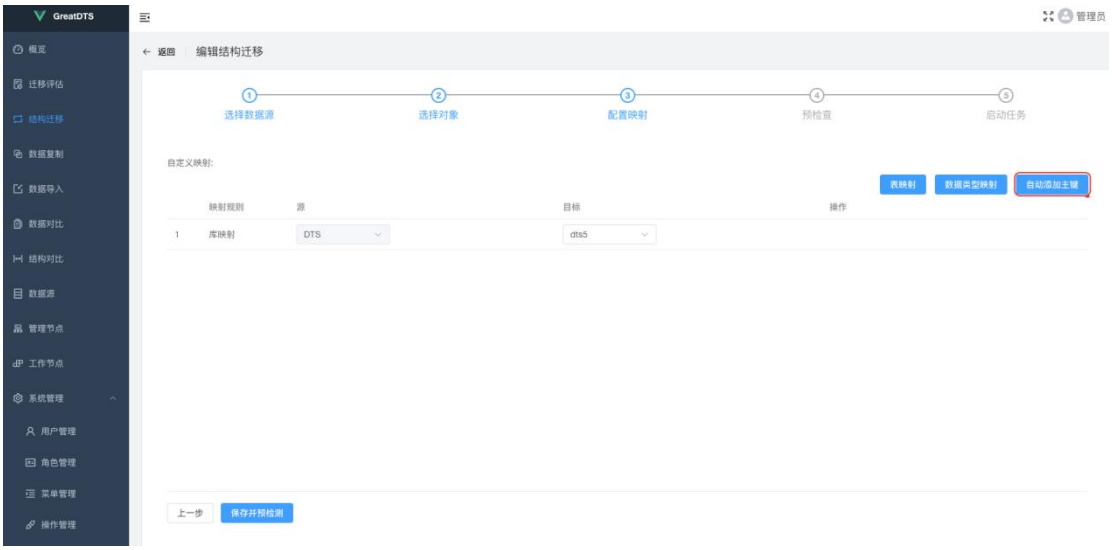


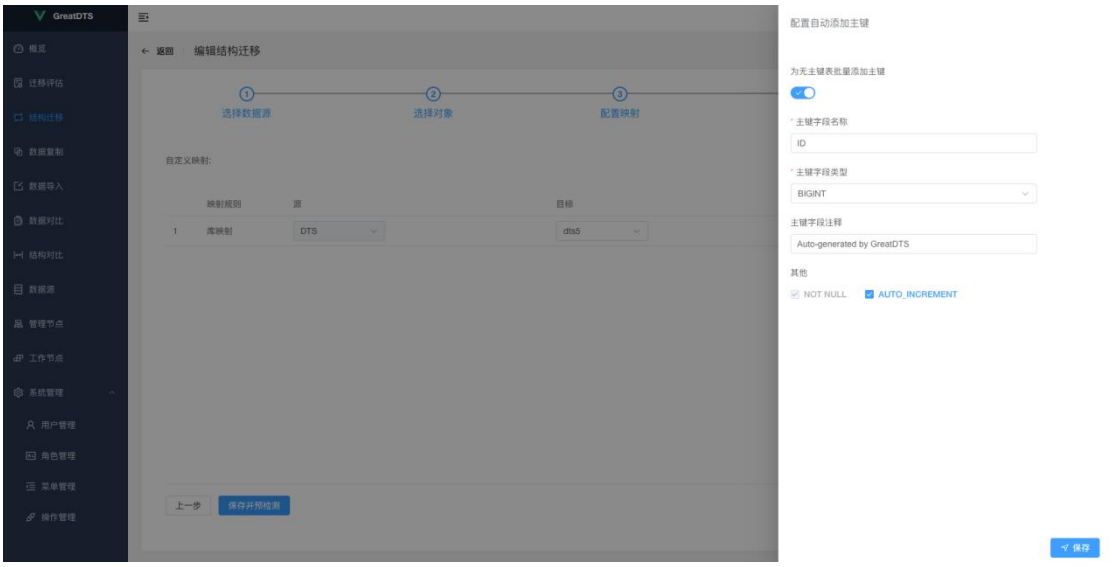




自动添加主键：源端为 oracle 时，源表中无主键，结构迁移时可以配置在写入目标端时自动添加主键。目标端支持 mysql 和 GreatDB 及 GreatDB OEM 的产品。

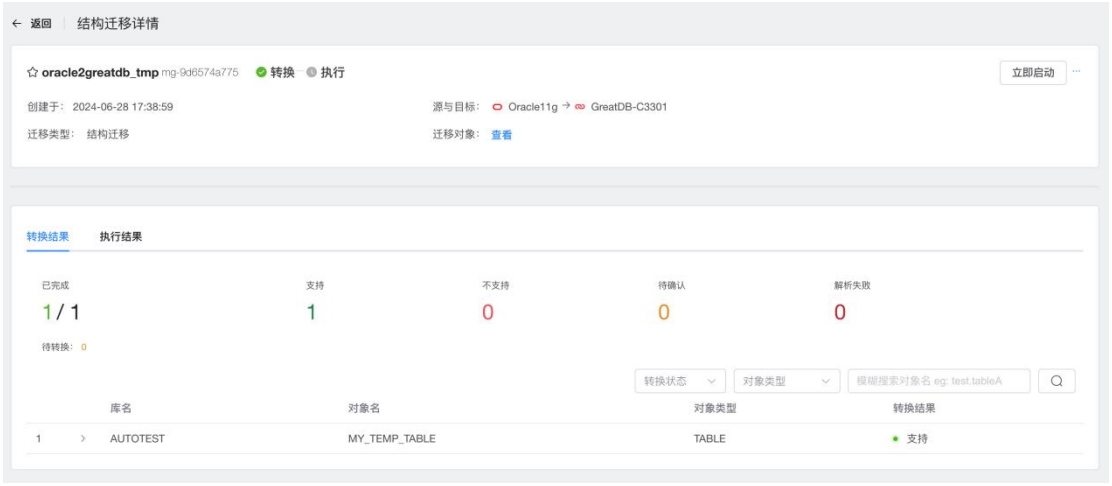
- 默认为无主键表自动添加主键的开关为关闭状态。
- 支持用户配置主键字段的名称
- 支持 INYINT、SMALLINT、MEDIUMINT、INT、BIGINT、CHAR、VARCHAR 数据类型
- 支持配置字段注释
- 支持选择是否启用自增（当数据类型选择 CHAR、VARCHAR 时，无法启用自增）





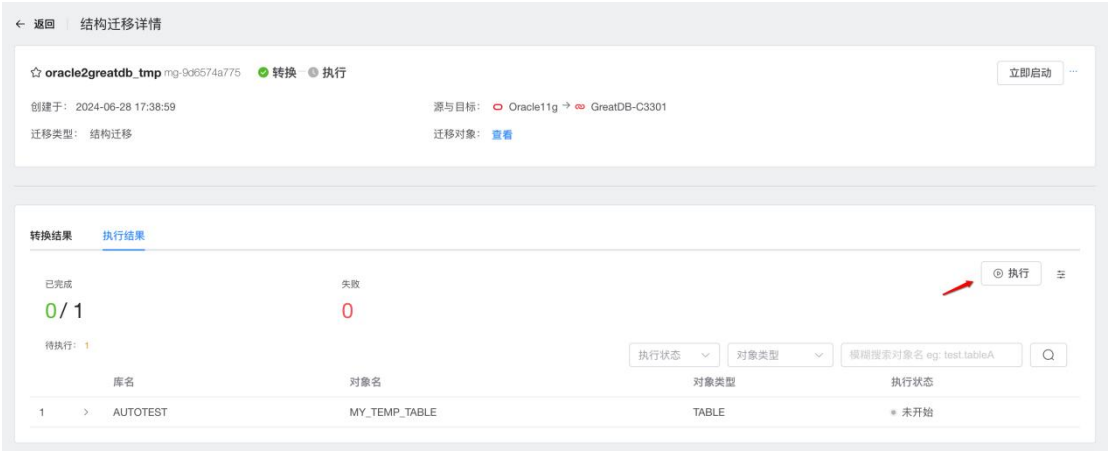
4.4.2 启动和执行

从列表页面或者详情页启动结构迁移。



切换到执行的结果页，展开行信息，用户可以进行二次编辑

点击执行结果标签页，右上部分【执行】按钮，向目标库执行所有转换后的 SQL



4.5 数据复制

GreatDTS 数据复制支持多种同构、异构数据源之间的离线、实时数据复制。适合数据迁移、数据库扩缩容、数据库版本升级、异地容灾、异地多活等多种业务场景。

4.5.1 基本流程

数据复制支持三种模式：

- 全量复制
- 增量复制
- 全量复制+增量复制

注意事项：

a. 在增量复制中，需要配置 DTS 的事务表（勾选增量复制时，弹出配置），事务表被创建在目标数据源中。事务表仅记录无主键表同步的事务信息，避免重复投递。

b. 全量复制+增量复制模式，若源数据源类型时 MySQL/GreatDB 且复制的对象存在无主键表，则应从业务上保证在全量复制阶段没有无主键表的数据写入，避免增量复制启动时，无主键表的数据重复

c. ORACLE 的增量复制，从此入口使用的是 LOGMNR 模式，此模式效率较低，且有诸多限制，若需要使用 ORACLE 的增量复制，请跳转至【数据导入】

4.5.2 创建数据复制

1. 数据库初始配置

使用数据复制需要对数据库做如下初始配置

Oracle11g 初始化

1）查看是否为归档日志模式：

```
Shell
archive log list
```

2）如果显示未启用归档日志模式，需要按照以下步骤启用：

```
Shell
#提示：以下操作会重启数据库，对数据库上的在线业务有影响，请安排在合适时间执行。
SQL> shutdown immediate
SQL> startup mount
SQL> alter database archivelog;
SQL> alter database open;
# 强制写日志
SQL> alter database force logging;
```

3）对库开启全列补充日志：

```
Shell
SQL> alter database add supplemental log data (all) columns;
```

4）查询并确认修改是否生效

```
Shell
SQL> select force_logging, supplemental_log_data_min,
supplemental_log_data_pk from v_$database;
```

5）为用户赋权，此用户为配置在数据源中的用户

```
Shell
CREATE ROLE logmnr_role;
GRANT CREATE SESSION TO logmnr_role;
GRANT SELECT ON V_$DATABASE to logmnr_role;
GRANT FLASHBACK ANY TABLE TO logmnr_role;
```

```
GRANT SELECT ANY TABLE TO logmnr_role;
GRANT SELECT_CATALOG_ROLE TO logmnr_role;
GRANT EXECUTE_CATALOG_ROLE TO logmnr_role;
GRANT SELECT ANY TRANSACTION TO logmnr_role;
GRANT SELECT ANY DICTIONARY TO logmnr_role;

GRANT CREATE TABLE TO logmnr_role;
GRANT LOCK ANY TABLE TO logmnr_role;
GRANT ALTER ANY TABLE TO logmnr_role;
GRANT CREATE SEQUENCE TO logmnr_role;

GRANT EXECUTE ON DBMS_LOGMNR TO logmnr_role;
GRANT EXECUTE ON DBMS_LOGMNR_D TO logmnr_role;

GRANT SELECT ON V_$LOG TO logmnr_role;
GRANT SELECT ON V_$LOG_HISTORY TO logmnr_role;
GRANT SELECT ON V_$LOGMNR_LOGS TO logmnr_role;
GRANT SELECT ON V_$LOGMNR_CONTENTS TO logmnr_role;
GRANT SELECT ON V_$LOGMNR_PARAMETERS TO logmnr_role;
GRANT SELECT ON V_$LOGFILE TO logmnr_role;
GRANT SELECT ON V_$ARCHIVED_LOG TO logmnr_role;
GRANT SELECT ON V_$ARCHIVE_DEST_STATUS TO logmnr_role;
```

6）创建用户

该用户提供给 DTS 用户读取增量日志日志，

```
Shell
# 请修改用户和密码
CREATE USER ${user_name} IDENTIFIED BY ${o_user_password};
GRANT CREATE SESSION TO ${user_name};
GRANT SELECT ANY TABLE TO ${user_name};
GRANT SELECT ANY DICTIONARY TO ${user_name};
GRANT SELECT_CATALOG_ROLE to ${user_name};
GRANT FLASHBACK ANY TABLE TO ${user_name};
GRANT logmnr_role to ${user_name};
ALTER USER ${user_name} quota unlimited on users;
```

Oracle19c + 初始化

1）设置闪回恢复区 db_recovery，归档日志保存在闪回恢复区

```
SQL
-- 闪回区路径请根据实际情况修改
$sqlplus / as sysdba
SQL> alter system set db_recovery_file_dest_size = 10G;
-- 以下目录需要提前创建和修改目前：
-- mkdir -p /opt/oracle/oradata/recovery_area
-- chmod -R 755  /opt/oracle/oradata/recovery_area
SQL> alter system set db_recovery_file_dest =
'/opt/oracle/oradata/recovery_area' scope=spfile;
```

2）查看是否为归档日志模式：

```
SQL
SQL> archive log list
```

3）如果显示未启用归档日志模式，需要按照以下步骤启用：

```
SQL
--提示：以下操作会重启数据库，对数据库上的在线业务有影响，请安排在合适时间执行。
SQL> shutdown immediate
SQL> startup mount
SQL> alter database archivelog;
SQL> alter database open;
-- 切换到CDB，强制写日志
SQL> alter session set container = CDB$ROOT;
SQL> alter database force logging;
```

4）对库开启全列补充日志：

```
Shell
SQL> alter database add supplemental log data (all) columns;
```

5）查询并确认修改是否生效

```
Shell
SQL> select force_logging, supplemental_log_data_min,
supplemental_log_data_pk from v$database;
```

6）创建用户

以下脚本中CDB（ORCLCDB）和PDB（ORCLPDB1）请根据实际情况修改

```Shell
# 关闭登录失败次数锁定限制
# ALTER PROFILE DEFAULT LIMIT FAILED_LOGIN_ATTEMPTS UNLIMITED;

# 查询 SUPPLEMENTAL Log 状态
# 检查是否开启最小补充日志
# SELECT supplemental_log_data_min FROM v$database;
```

7）为 PDB 开启附加日志

```Shell
$ sqlplus sys/root123@//localhost:1521/ORCLPDB1 as sysdba
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
```

8）创建 TABLESPACE

```SQL
-- Create Log Miner Tablespace and User
$ sqlplus sys/root123@//localhost:1521/ORCLCDB as sysdba
 CREATE TABLESPACE LOGMINER_TBS DATAFILE
'/opt/oracle/oradata/ORCLCDB/logminer_tbs.dbf' SIZE 25M REUSE
AUTOEXTEND ON MAXSIZE UNLIMITED;

 $ sqlplus sys/root123@//localhost:1521/ORCLPDB1 as sysdba
 CREATE TABLESPACE LOGMINER_TBS DATAFILE
'/opt/oracle/oradata/ORCLCDB/ORCLPDB1/logminer_tbs.dbf' SIZE 25M
REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

9）创建用户并授权

```SQL
-- 请按需修改以下语句中的用户和密码(c##sync)密码<PASSWORD>
$ sqlplus sys/root123@//localhost:1521/ORCLCDB as sysdba

CREATE USER c##sync IDENTIFIED BY <PASSWORD> DEFAULT TABLESPACE
LOGMINER_TBS QUOTA UNLIMITED ON LOGMINER_TBS CONTAINER=ALL;

GRANT CREATE SESSION TO c##sync CONTAINER=ALL;
GRANT SET CONTAINER TO c##sync CONTAINER=ALL;
GRANT SELECT ANY DICTIONARY TO c##sync CONTAINER=ALL;
```

```
GRANT SELECT ON V_$DATABASE to c##sync CONTAINER=ALL;
GRANT FLASHBACK ANY TABLE TO c##sync CONTAINER=ALL;
GRANT SELECT ANY TABLE TO c##sync CONTAINER=ALL;
GRANT SELECT_CATALOG_ROLE TO c##sync CONTAINER=ALL;
GRANT EXECUTE_CATALOG_ROLE TO c##sync CONTAINER=ALL;
GRANT SELECT ANY TRANSACTION TO c##sync CONTAINER=ALL;
GRANT LOGMINING TO c##sync CONTAINER=ALL;

GRANT CREATE TABLE TO c##sync CONTAINER=ALL;
GRANT LOCK ANY TABLE TO c##sync CONTAINER=ALL;
GRANT ALTER ANY TABLE TO c##sync CONTAINER=ALL;
GRANT CREATE SEQUENCE TO c##sync CONTAINER=ALL;

GRANT EXECUTE ON DBMS_LOGMNR TO c##sync CONTAINER=ALL;
GRANT EXECUTE ON DBMS_LOGMNR_D TO c##sync CONTAINER=ALL;

GRANT SELECT ON V_$LOG TO c##sync CONTAINER=ALL;
GRANT SELECT ON V_$LOG_HISTORY TO c##sync CONTAINER=ALL;
GRANT SELECT ON V_$LOGMNR_LOGS TO c##sync CONTAINER=ALL;
GRANT SELECT ON V_$LOGMNR_CONTENTS TO c##sync CONTAINER=ALL;
GRANT SELECT ON V_$LOGMNR_PARAMETERS TO c##sync CONTAINER=ALL;
GRANT SELECT ON V_$LOGFILE TO c##sync CONTAINER=ALL;
GRANT SELECT ON V_$ARCHIVED_LOG TO c##sync CONTAINER=ALL;
GRANT SELECT ON V_$ARCHIVE_DEST_STATUS TO c##sync CONTAINER=ALL;
```

10）为单独的表开启归档日志

为开启归档日志之前创建的表单独设置开启归档日志

```
SQL
-- 配置表开启 SUPPLEMENTAL LOG
ALTER TABLE customers ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
```

在增量配置页面，选择使用 LogReplicator 方式选择完成增量日志复制时需配置如下权限。

```
SQL
GRANT SELECT, FLASHBACK ON SYS.CCOL$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.CDEF$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.COL$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.DEFERRED_STG$ TO <USER>;
```

```
GRANT SELECT, FLASHBACK ON SYS.ECOL$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.LOB$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.LOBCOMPPART$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.LOBFRAG$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.OBJ$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.TAB$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.TABCOMPART$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.TABPART$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.TABSUBPART$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.TS$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.USER$ TO <USER>;
GRANT SELECT ON SYS.IND$ TO <USER>;
GRANT SELECT ON SYS.ICOL$ TO <USER>;
GRANT SELECT ON SYS.V_$ARCHIVED_LOG TO <USER>;
GRANT SELECT ON SYS.V_$DATABASE TO <USER>;
GRANT SELECT ON SYS.V_$DATABASE_INCARNATION TO <USER>;
GRANT SELECT ON SYS.V_$LOG TO <USER>;
GRANT SELECT ON SYS.V_$LOGFILE TO <USER>;
GRANT SELECT ON SYS.V_$PARAMETER TO <USER>;
GRANT SELECT ON SYS.V_$STANDBY_LOG TO <USER>;
GRANT SELECT ON SYS.V_$TRANSPORTABLE_PLATFORM TO <USER>;
```

GreatDB/MySQL 初始化

1）开启 binlog，配置 binlog-format

```
sql
-- 为 GreatDB/MySQL 开启 binlog，并配置 binlog-format = ROW， my.cnf 配置如下

[mysqld]
log-bin=mysql-bin # 开启 binlog
binlog-format=ROW # 选择 ROW 模式



> show variables like "log_bin";
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| log_bin       | ON    |
+---------------+-------+
```

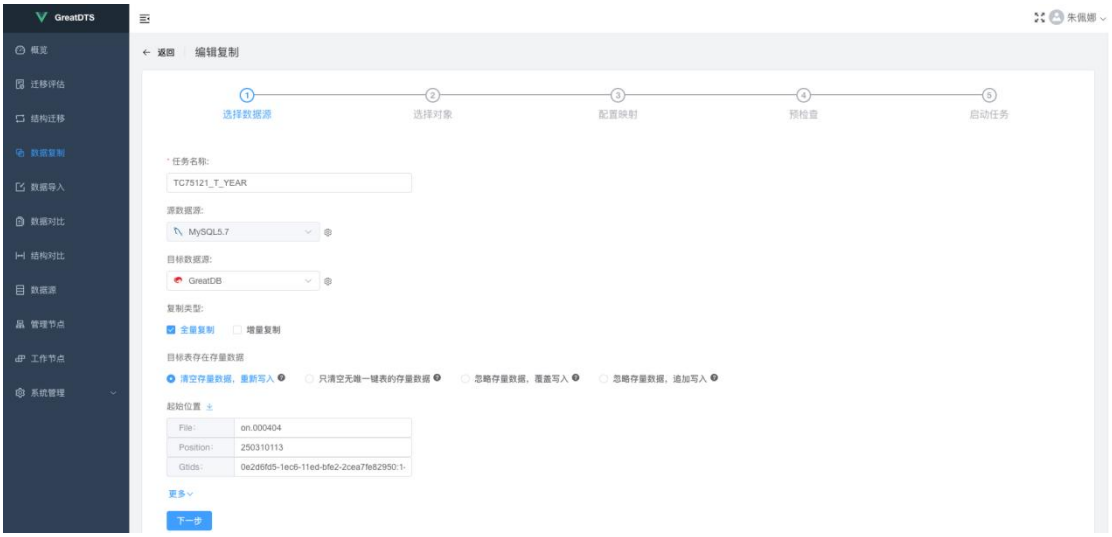
```
> show variables like 'binlog_format';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| binlog_format | ROW   |
+---------------+-------+
1 row in set (0.00 sec)
```

2）授权 slave 权限，此用户为数据源中配置的用户

```
sql
CREATE USER '<user>'@'%' IDENTIFIED BY '<PASSWORD>';
GRANT SELECT, REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO
'<user>'@'%';
FLUSH PRIVILEGES;
```

2. 选择数据源

在选择源数据源时，系统会自动获取数据源的 offset（oracle 为 scn 、greatdb/mysql 为 gtid），用户可以再此编辑。



打开“更多”，配置项中，可以指定全量任务表的并行度，缓冲区大小等。

参数	说明
源数据源	源数据源
目标数据源	目标数据源
复制类型	全量复制 增量复制
目标表存在存量数据的处理策略	• 清空存量数据，重新写入：任务首次启动时，执行清空表操作（truncate table），并使用 insert 插入数据，当任务暂停或异常停止后再次启动，未完成的无唯一键表会再次 truncate 重新写入，有唯一键的表则会保留表数据，使用 merge 继续写入 • 只清空无唯一键表的存量数据：仅清空无唯一键表的数据，有唯一键的表数据则会保留，使用 merge 覆盖写入 • 忽略存量数据，覆盖写入：不对目标表做初始化行为，无唯一键的表采用追加写入，有唯一键的表采用 merge 覆盖写入 • 忽略存量数据，追加写入：不对目标表做初始化行为，表将都采用 INSERT 写入（若有唯一键表有数据，则可能出现唯一键冲突）
起始位置	• oracle 作为源 全量阶段，开启 use flashback 会使用用户填写的指定的

	SCN 号，读取最全量数据，同时作为增量的起点 全量阶段，关闭 use flashback，无论用户是否填写 SCN 号都使用任务启动时数据库最新的 SCN 号，读取全量数据，同时作为增量的起点 • Mysql or greatdb 作为源 无论是否填写日志位置，都会使用最新的日志位置读取全量数据 用户填写或点击起始位置旁边的按钮获取当前数据源的最新日志位置作为增量的起点
并行的表数量	一个任务同时并行的表的数量
单表的查询线程数	一个任务中同单表的并行数
检查点间隔	强制同步从 DTS 数据缓存区刷入目标库的最大间隔
缓冲区容量	缓存区的最大数据行数，超过此数值，强制同步从 DTS 数据缓存区刷入目标库
缓冲区容量	缓存区的最大容量，，超过此数值，强制同步从 DTS 数据缓存区刷入目标库
限流	限制每秒从源数据源读取的行数。默认 0 不限制。当设置限流时，写入并行度将被设置为 1。即有限流就没有并行度。
批提取数据大小	每次从读缓冲区提取的最大数据行数
写入批处理大小	批量写入目标端大小
写入最大间隔	全量阶段，写入目标最大间隔
写入失败重试次数	写入目标端失败时的重试次数
写入并行度	写入目标端线程数
增量无主键表写	增量无主键表写入目标端的并行度

入并行度	
源库 string 类型的强制转换编码	指定 string 类型的强制转换编码。不支持 NCLOB。举例：源库字符集是 US7ASCII，而写入的编码是 UTF-8，则需要设置此属性
JDBC 空写	配置开启或关闭 JDBC 空写，当开启 JDBC 空写时，仅从源端读取数据，不对目标端做写入，适用于测试源端读取速度的场景。

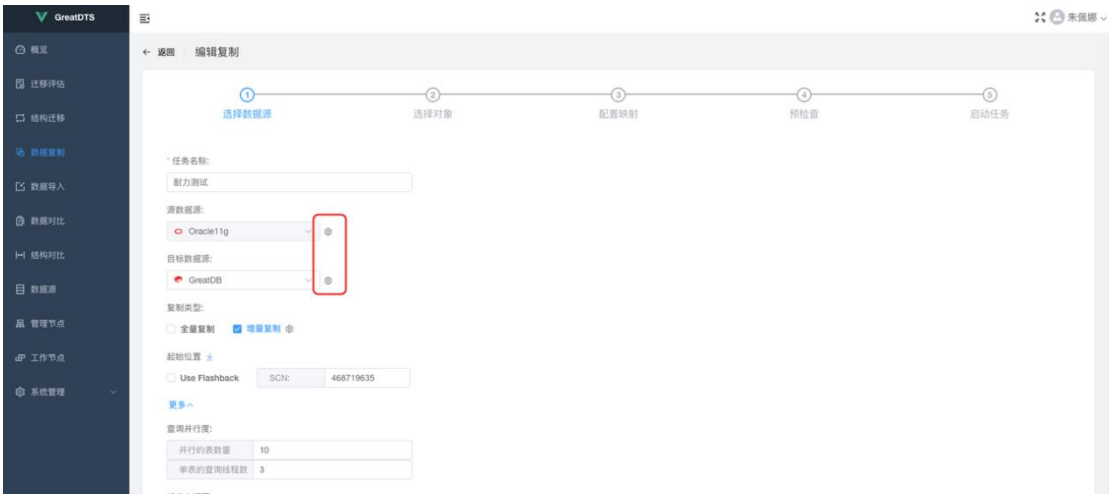
Environment Props 参数说明

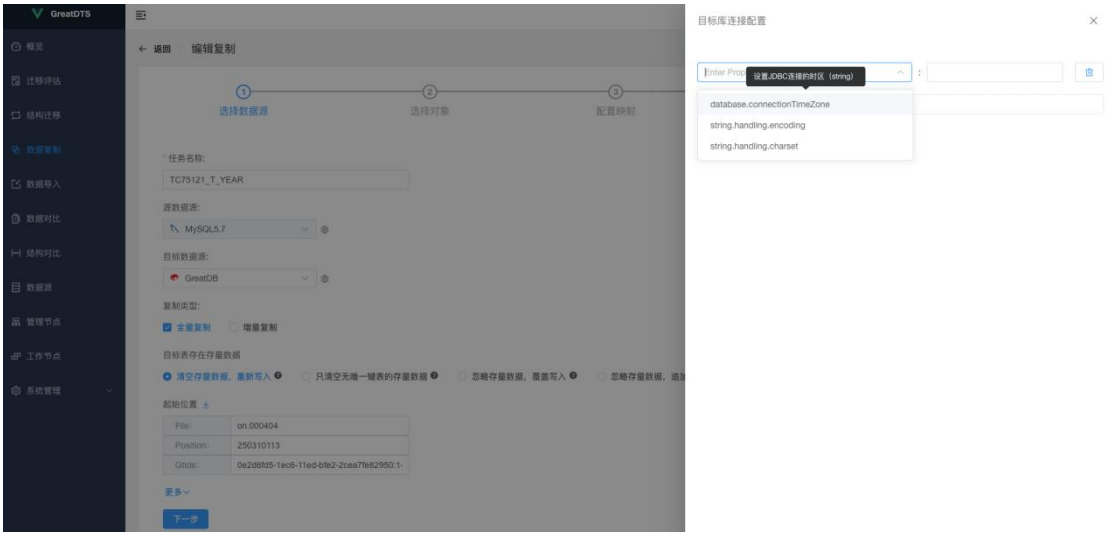
参数	说明	值域范围	默认值
execution.checkpointing.timeout	检查点允许的最大超时间 (s)（number）	正整数	60 单位秒
restart.strategy	故障恢复策略（string）	none off disable fixed-delayfailure-rate	none
restart-strategy.fixed-delay.attempts	故障恢复次数（number）	正整数	3
restart-strategy.fixed-delay.delay	故障恢复延迟（ms）（number） 发生故障时，等待指定时间后，尝试恢复，使用场景；当数据库的延迟比较高，或者机器性能比较差，需要 dts 遇到故障以后，多等久点再尝试恢复	正整数	5000 单位毫秒

Transfommer Props 参数说明

参数	说明	值域范围	默认值
ddl.convert.enabled	开启转换 DDL，会重写源库传入的 DDL（string） 适合同构数据库间做灾备场景，将此参数设置为 false 实现 DDL 原样的透传	true、false	true

点击源端和目标端数据源右侧的齿轮可以配置源和目标端的 url 配置，可以从列表中选择需要配置的项，也可以直接填写需要配置的项，如果填写的配置不存在，不会对复制造成影响。





源端配置项说明：

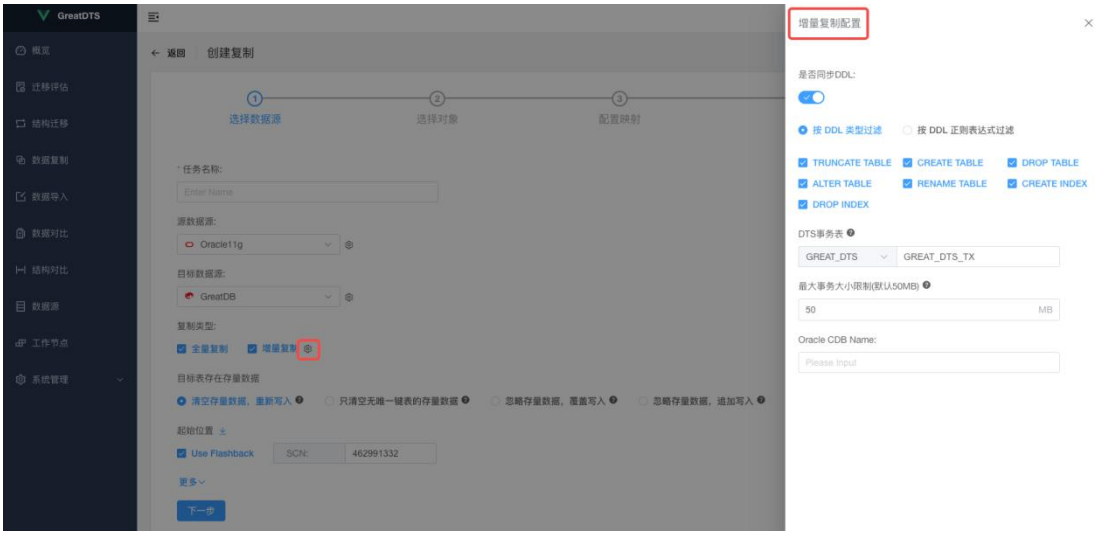
database.connectionTimeZone	指定 JDBC 连接的时区（string）

string.handling.encoding	ORACLE 源库 string 类型的强制转换编码，指定 string 类型的强制转换编码。不支持 NCLOB。举例：源库字符集是 US7ASCII，而写入的编码是 UTF-8，则需要设置此属性（string）
connection.keep.alive	使用单独的线程来保证 binlog dump 连接（boolean）

目标端配置项说明

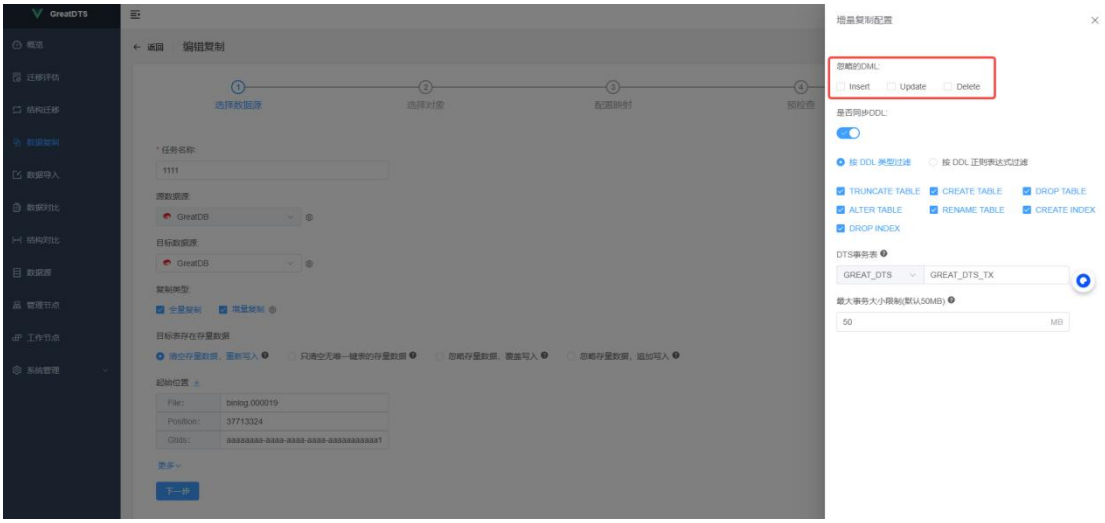
database.connectionTimeZone	指定 JDBC 连接的时区（string）
string.handling.encoding	ORACLE 目标库写入的编码（string）
string.handling.charset	ORACLE 目标库的字符集（string）
initial.sqls	使用 JDBC 创建连接后执行的初始化 SQL 语句，多个 SQL 使用分号分割。（string）

- 点击“增量复制”右侧齿轮可以对增量复制做配置



配置忽略的 DML

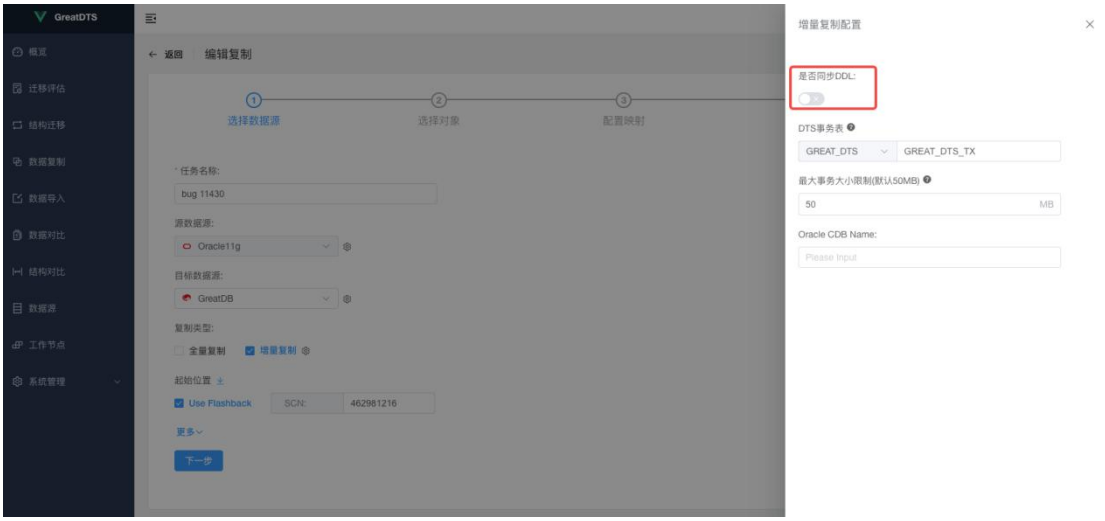
可以选择配置忽略 dml，支持选择 insert、update、delete 中的一种或者多种

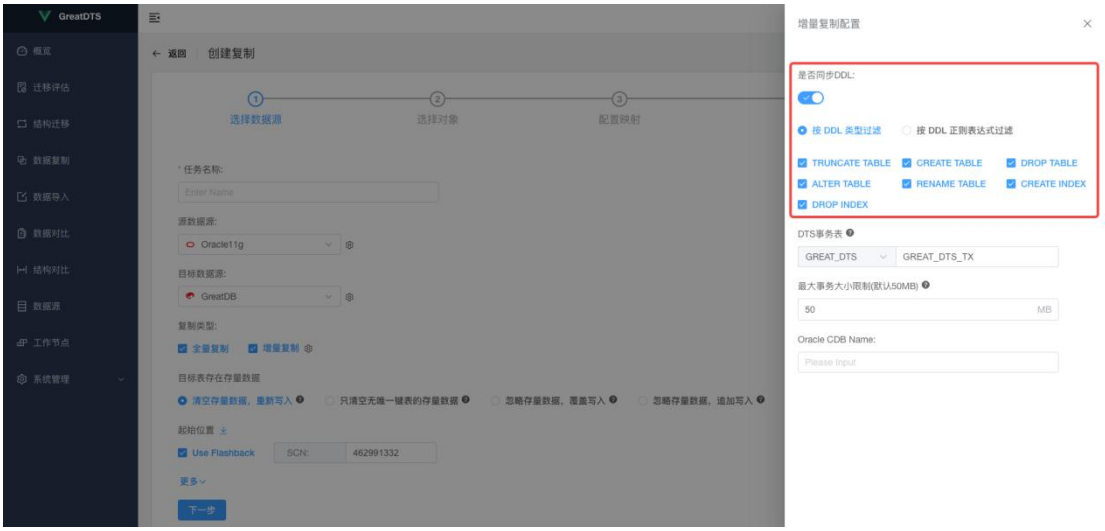


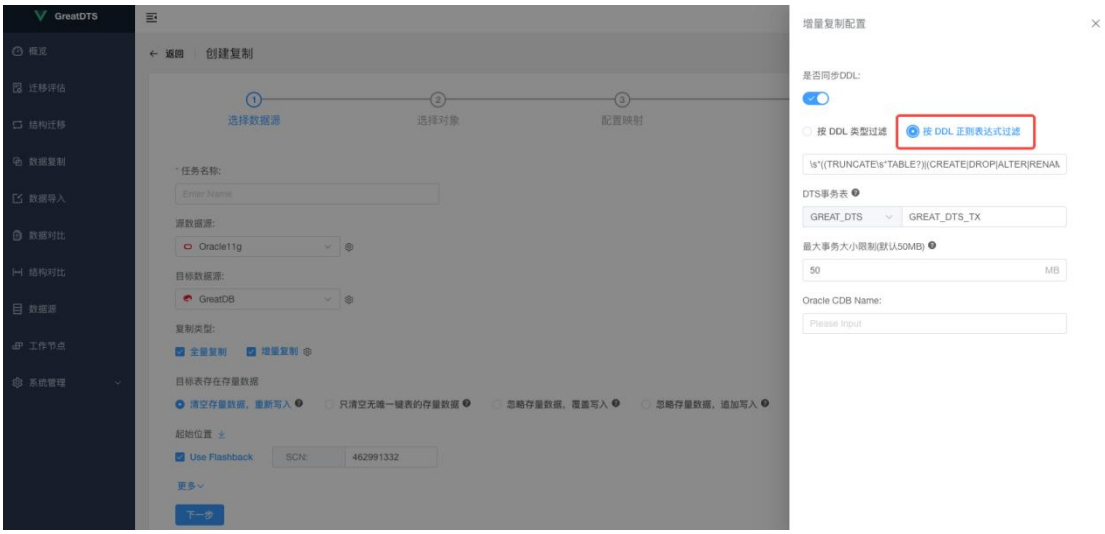
DDL 同步（此功能为 611-GA3 的新增功能）

从 6.1.1-GA3 版本开始在增量复制环节支持 DDL，可以通过勾选或正则完成对 DDL 同步的配置。默认 DDL 的同步为开开启状态。

1）点击是否同步 DDL 可以开启或关闭同步 DDL。







2）DDL 类型过滤

按 DDL 类型过滤，可以勾选需要同步的 DDL 类型如上图所示。

按 DDL 正则表达式过滤，如下为同步所有支持的 DDL 的示例，如需要调整，可根据正则表达式的语法自行调整

```
Plain Text
\s*((TRUNCATE\s*TABLE?)|(CREATE|DROP|ALTER|RENAME)\s*(TABLE|INDEX))(?s:.*)
```

3）源和目标端 DDL 同步支持情况

源	目标	支持的 DDL	版本说明
Oracle	GreatDB/MySQL	truncate table the_table	6.1.1-GA3（Logminer 支持，OLR 暂不支

			持）
		create table the_table (...)	6.1.1-GA3（Logminer 支持，OLR 暂不支持）
		drop table the_table	6.1.1-GA3（Logminer 支持，OLR 暂不支持）
		alter table the_table rename to the_table1	6.1.1-GA3（Logminer 支持，OLR 暂不支持）
		alter table the_table add primary key (col1, col2, ...)	6.1.1-GA3（Logminer 支持，OLR 暂不支持）
		alter table the_table add constraint pk primary key (col1, col2, ...)	6.1.1-GA3（Logminer 支持，OLR 暂不支持）
		alter table the_table drop primary key	6.1.1-GA3（Logminer 支持，OLR 暂不支持）
		alter table the_table add constraint uk unique (col1, col2, ...)	6.1.1-GA3（Logminer 支持，OLR 暂不支持）
		alter table the_table add (col1 xxx, col2 xxx)	6.1.1-GA3（Logminer 支持，OLR 暂不支

			持）
		alter table the_table modify (col1 xxx, col2 xxx)	6.1.1-GA3（Logminer 支持，OLR 暂不支持）
		alter table the_table drop column col1	6.1.1-GA3（Logminer 支持，OLR 暂不支持）
		create index the_index on the_table(col1, col2, ...)	6.1.1-GA3（Logminer 支持，OLR 暂不支持）
		drop index the_index	6.1.1-GA3（Logminer 支持，OLR 暂不支持）
MySQL	Oracle	truncate table the_table	6.1.1-GA3
		truncate the_table	6.1.1-GA3
		create table the_table (....)	6.1.1-GA3
		drop table the_table	6.1.1-GA3
		alter table the_table1 rename to the_table2	6.1.1-GA3
		alter table add column col1 xxx	6.1.1-GA3
		alter table the_table drop col1	6.1.1-GA3
		alter table the_table add primary key (col1, col2, ...)	6.1.1-GA3

		alter table the_table drop primary key	6.1.1-GA3
		alter table the_table add index index_name(col1, col2, ...)	6.1.1-GA3
		alter table the_table add unique index index_name(col1, col2, ...)	6.1.1-GA3
		alter table the_table drop index unique_index_name;	6.1.1-GA3
GreatDB/MySQL	GreatDB/MySQL	truncate table the_table	6.1.1-GA3
		truncate the_table	6.1.1-GA3
		create table the_table (....)	6.1.1-GA3
		drop table the_table	6.1.1-GA3
		rename table table_a to table_b	6.1.1-GA3
		alter table the_table1 rename to the_table2	6.1.1-GA3
		alter table add column col1 xxx	6.1.1-GA3
		alter table the_table drop col1	6.1.1-GA3
		alter table the_table add primary key (col1, col2, ...)	6.1.1-GA3
		alter table the_table drop primary key	6.1.1-GA3
		alter table the_table add index index_name(col1, col2, ...)	6.1.1-GA3
		alter table the_table add unique	6.1.1-GA3

		index index_name(col1, col2, ...)	
		alter table the_table drop index unique_index_name;	6.1.1-GA3
		create index/unique index index_name on the_table(col1, col2, ...)	6.1.1-GA3
		drop index index_name on the_table	6.1.1-GA3

4）异常处理

场景一：遇到不在支持范围内的 DDL

处理方式：GreatDTS 不做处理，DDL 会被忽略

任务状态：任务正常运行不受影响

日志：被忽略的 DDL 会被打印至日志

解决方案：无需处理

场景二：在支持范围内的 DDL 遇到执行报错的情况

处理方式：GreatDTS 报错会通过日志输出，任务将停止

任务状态：任务停止

日志：记录报错日志，DDL 会被打印至日志

解决方式：排查报错日志，人工决策是否手动在目标端执行。决策或执行完成后，修改任务的增量配置跳过此类型 DDL，最后重新启动任务。任务正常后可以根据情况选择是否要再次开启此类型的 DDL。

注：GreatDTS 的任务配置需要重启任务才会生效。

DTS 事务

1）配置 DTS 事务表

用于记录无主键表增量同步过程中事务提交时的相关信息

2）最大事务大小限制(默认 50MB)

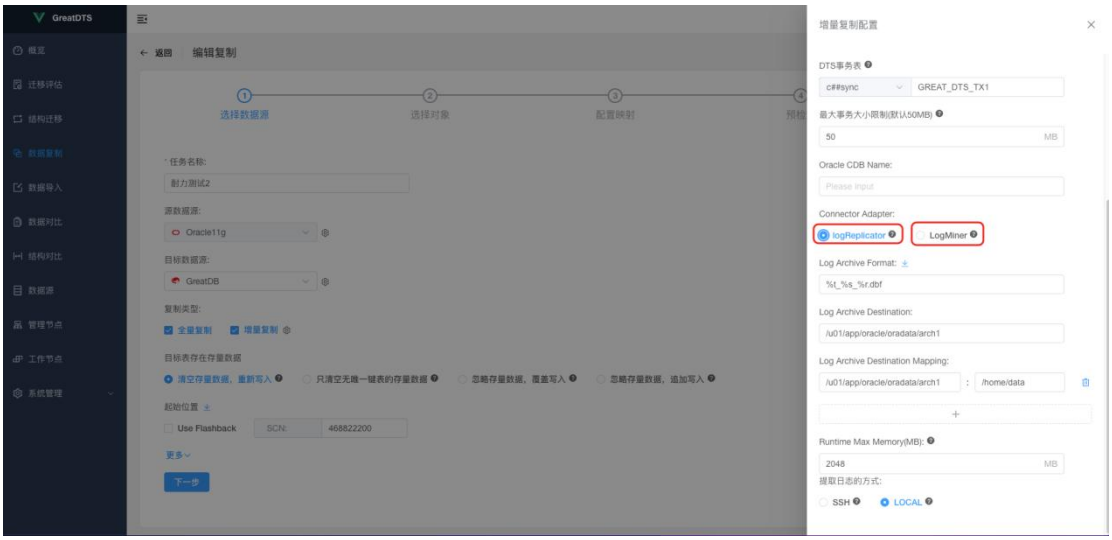
超过此限制的事务，将被拆分成多个事务提交

Oracle CDB Name

仅 oracle 支持

Oracle 增量复制

620 版本之后新增功能，可以在页面上可视化的配置 oracle 的增量配置，可以替代左侧菜单的数据导入功能。支持 1. logreplcator 直接解析归档日志文件获取增量数据；2.logminer 通过 Oracle LogMiner 获取在线/归档日志获取增量数据。当选择“直接解析归档日志文件获取增量数据”需要配置日志格式和日志位置等信息，后文中详细介绍。

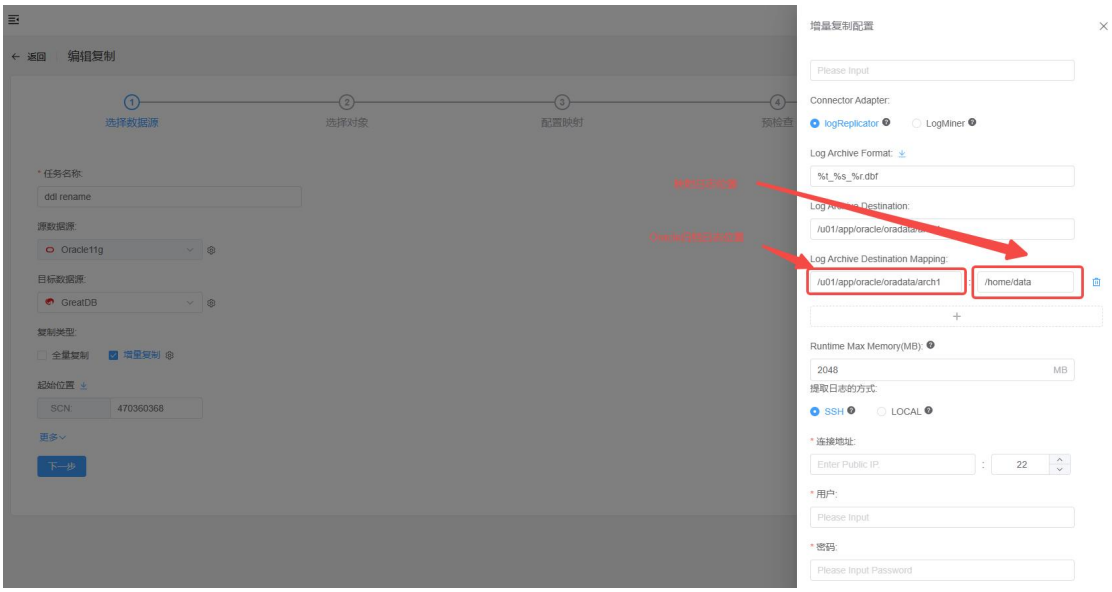


logreplcator 直接解析归档日志文件获取增量数据方式的参数介绍

名称	说明	数据填充方式
Log Archive Format	oracle 日志格式	1. 用户手填 2. 点击 Log Archive Format 右侧的按钮会自动获取 如果获取的不准确需要用户手动校对一下
Log Archive Destination	oracle 日志目录	1. 用户手填 2. 点击 Log Archive Format 右侧的按钮会自动获取
Log Archive Destination Mapping	oracle 日志目录与拷贝到的 worker 主机的目录的对应位置	1. 用户手填 2. 点击 Log Archive Format 右侧的按钮会自动获取 可填多条，但是必须有一条，不然

		任务启动会报错。 关于 Log Archive Destination Mapping 配置，详见配置讲解
Runtime Max Memory(MB)	用于日志解析器缓存的未提交事务，如果日志中包含太多的未提交事务，需要调大此项	用户填写，默认值为 2048MB
提取方式	提取 oracle 日志的方式	方式一：通过 SSH 从远程服务器读取 需要用户手填 oracle 日志所在的主机信息：IP、SSH 端口、用户名、密码 方式二：从"Worker"宿主机的目录读取 根据任务详情页面配置派发策略选择对应的 worker

- Log Archive Destination Mapping 配置讲解



当日志提取方式为 SSH 从远程服务器读取时，分两种情况。

情况一：远程服务器为 Oracle 数据库实例所在的服务器。DTS 回去冒号右侧映射的目录去取日志。

如果是 11G 的数据库，用户需要手动添加两层目录 ./archivelog/archive/，并将数据库归档日志的目录位置与映射位置创建软连接。实现 DTS 实时日志读取。

如果是 19C 的数据库，用户需要手动添加两层目录 ./archivelog/，并将数据库归档日志的目录位置与映射位置创建软连接。实现 DTS 实时日志读取。

11G 示例：Oracle 的日志位置 /u01/app/oracle/oradata/arch1/ 填写的映射目录：/u01/app/oracle/oradata/

1）需要创建/u01/app/oracle/oradata/archivelog/archive/目录。

2）创建/u01/app/oracle/oradata/archivelog/archive/ 到 /u01/app/oracle/oradata/arch1/的软连接。

19C 示例：Oracle 的日志位置 /u01/app/oracle/oradata/arch1/ 填写的映射目录：/u01/app/oracle/oradata/

1）需要创建/u01/app/oracle/oradata/archivelog/目录。

2）创建/u01/app/oracle/oradata/archivelog/ 到 /u01/app/oracle/oradata/arch1/的软连接。

情况二：远程服务器为非 Oracle 数据库实例所在的服务器也非任务所使用的 worker 所在的服务器的其他主机。DTS 会去冒号右侧映射的目录去取日志。

如果是 11G 的数据库，用户需要手动添加两层目录 ./archivelog/archive/，并将日志实时拷贝至映射目录实现 DTS 实时日志读取。

如果是 19C 的数据库，用户需要手动添加一层目录 ./archivelog/，并将日志实时拷贝至映射目录实现 DTS 实时日志读取。

11G 示例：Oracle 的日志位置 /u01/app/oracle/oradata/arch1/ 填写的映射目录：/u01/app/oracle/oradata/

1）需要创建/u01/app/oracle/oradata/archivelog/archive/目录。

2）写一个自动拷贝日志的脚本实时将日志从 /u01/app/oracle/oradata/arch1/ 拷贝到远端的 /u01/app/oracle/oradata/archivelog/archive/目录。

19C 示例：Oracle 的日志位置 /u01/app/oracle/oradata/arch1/ 填写的映射目录：/u01/app/oracle/oradata/

1）需要创建/u01/app/oracle/oradata/archivelog/目录。

2）写一个自动拷贝日志的脚本实时将日志从 /u01/app/oracle/oradata/arch1/ 拷贝到远端的 /u01/app/oracle/oradata/archivelog/目录。

当日志提取方式为 LOCAL 服务器读取时

即远程服务器为任务所使用的 worker 节点所在的服务器。DTS 会去冒号右侧映射的目

录去取日志。

11G 示例：Oracle 的日志位置 /u01/app/oracle/oradata/arch1/ 填写的映射目录：/u01/app/oracle/oradata/

1）需要创建/u01/app/oracle/oradata/archivelog/archive/目录。

2）写一个自动拷贝日志的脚本实时将日志从 /u01/app/oracle/oradata/arch1/ 拷贝到远端的 worker 系节点 /u01/app/oracle/oradata/archivelog/archive/目录。

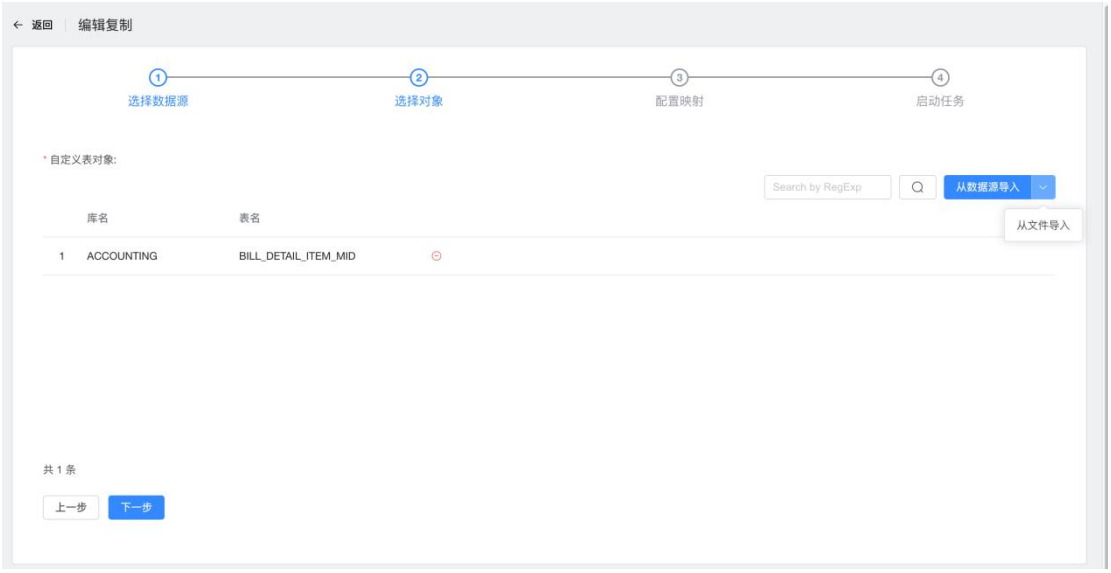
19C 示例：Oracle 的日志位置 /u01/app/oracle/oradata/arch1/ 填写的映射目录：/u01/app/oracle/oradata/

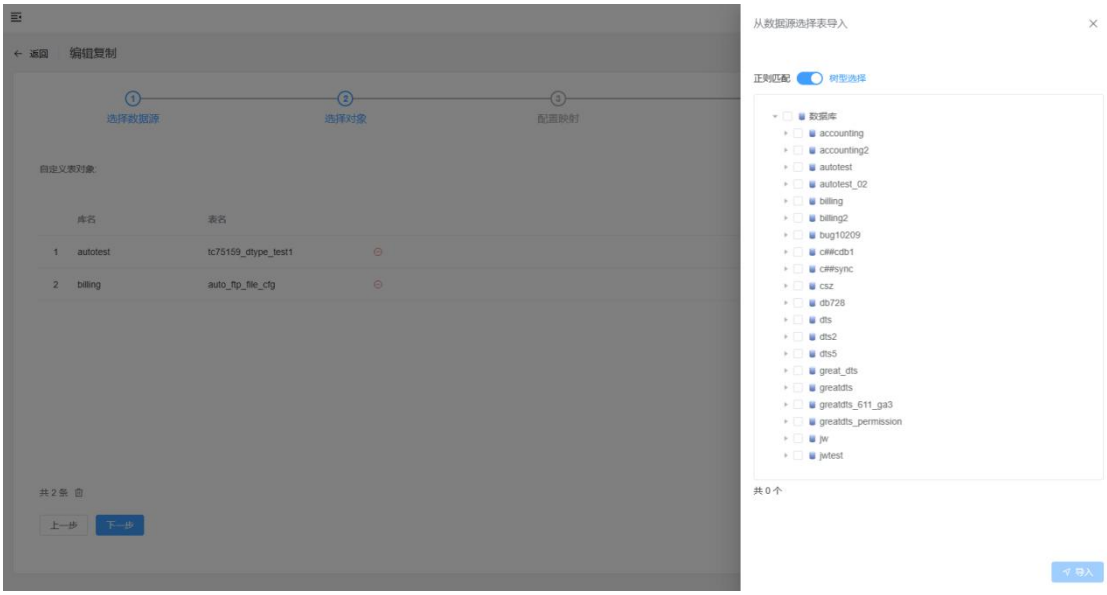
1）需要创建/u01/app/oracle/oradata/archivelog/目录。

2）写一个自动拷贝日志的脚本实时将日志从 /u01/app/oracle/oradata/arch1/ 拷贝到远端的 worker 系节点 /u01/app/oracle/oradata/archivelog/目录。

3. 选择对象

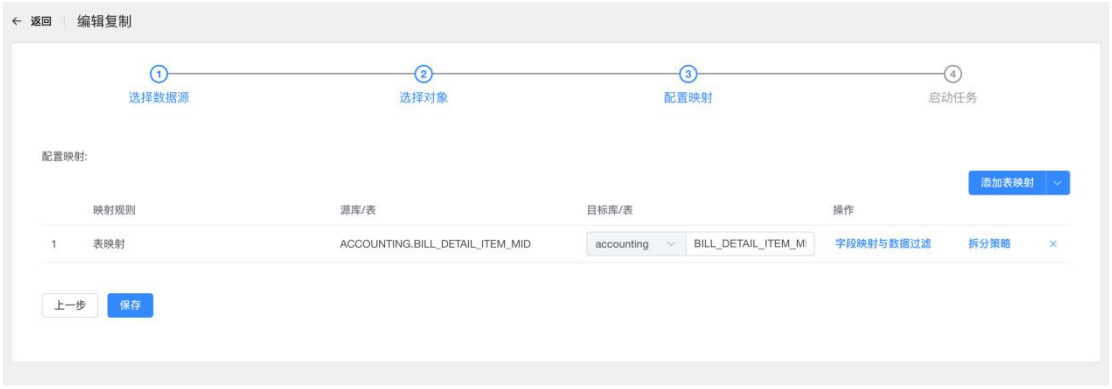
选择数据复制的对象：对象可以从源数据源导入（从 620 版本开始支持在页面上直接选择对象，也支持正则表达式匹配对象）或者从 excel 文件导入。





4. 配置映射

- 字段映射与数据过滤



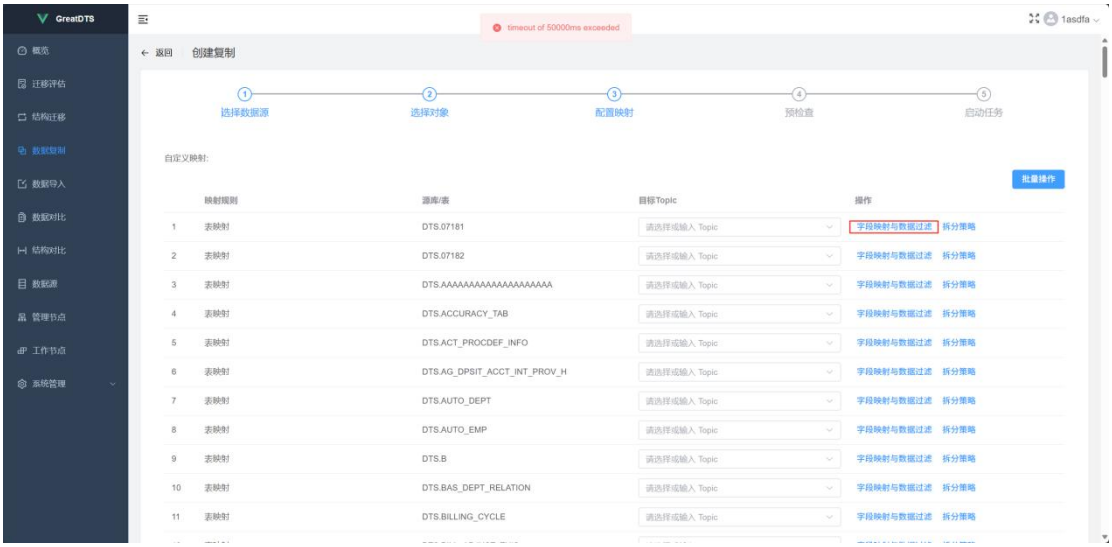
1）当源数据库和目标数据库存在表名称、列名称不同时，可以指定映射规则，

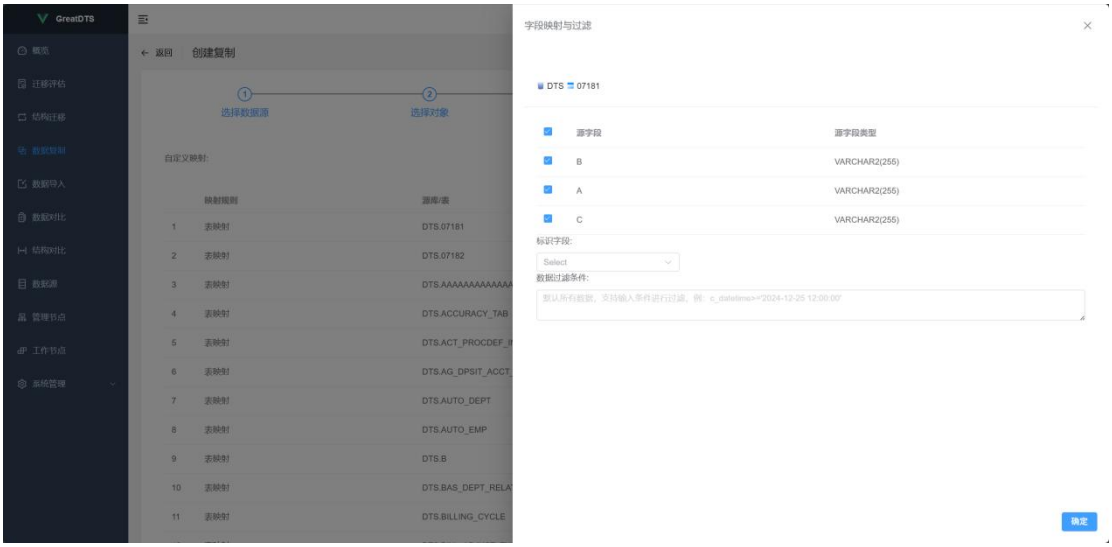
2）指定表的列范围和对比的数据集范围

3）指定标示字段，默认情况下 DTS 会自动识别表主键或唯一键作为标示字段，用户也可直接指定

4）不配置映射规则，默认表名、列名完成相同、数据集范围为全表

5）从 620-GA1 版本开始，目标端为 kafka 时也支持配置字段映射和数据过滤，替代了复杂的模板配置





- 添加表映射

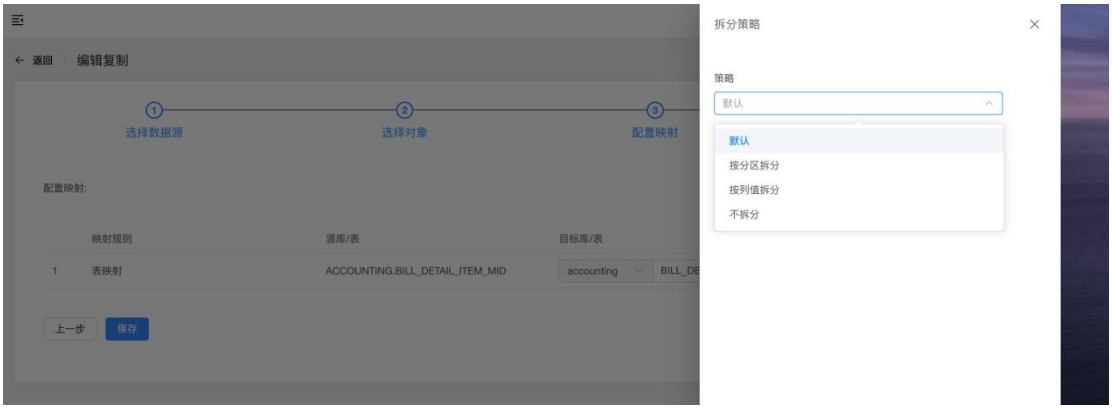
从右上角按钮【添加表映射】点击，在弹出的列表中勾选需要添加映射的表

- 添加表映射

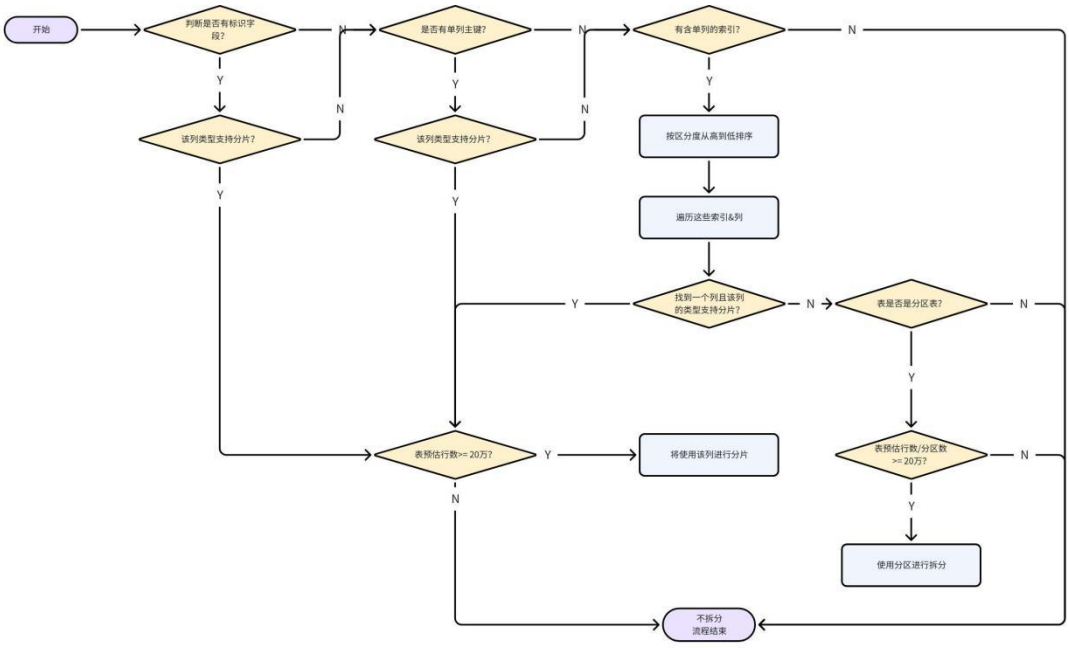
从右上角按钮【添加表映射】旁边的展开号点击【添加库映射】，在弹出的列表中勾选需要添加映射的库

- 拆分策略

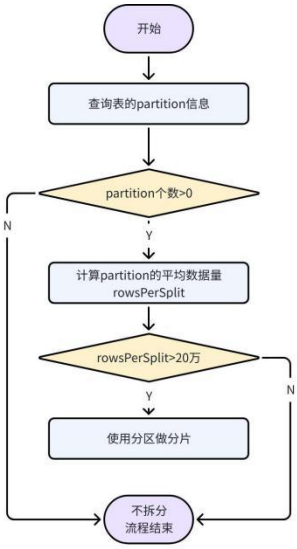
允许用户指定全量复制时，数据拆分策略



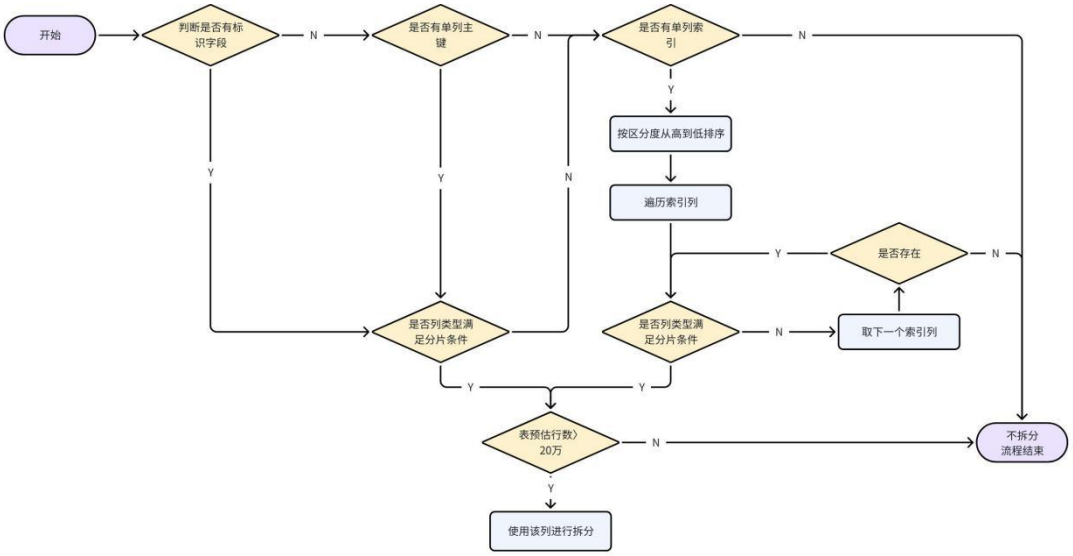
- 默认
  - 拆分条件：表预估数据量或表分区平均预估量>20 万
  - 拆分粒度：每 20 万条做一次拆分
  - 拆分策略：当有主键或索引时，自动优先选择离散度高索引拆分，有分区的按分区继续拆分，详细的判断流程如下



- 按分区拆分
  - 拆分条件：分区数量>0；分区内平均预估数据量>20 万
  - 拆分粒度：每 20 万条做一次拆分
  - 拆分策略：按分区拆分拆分流程如下



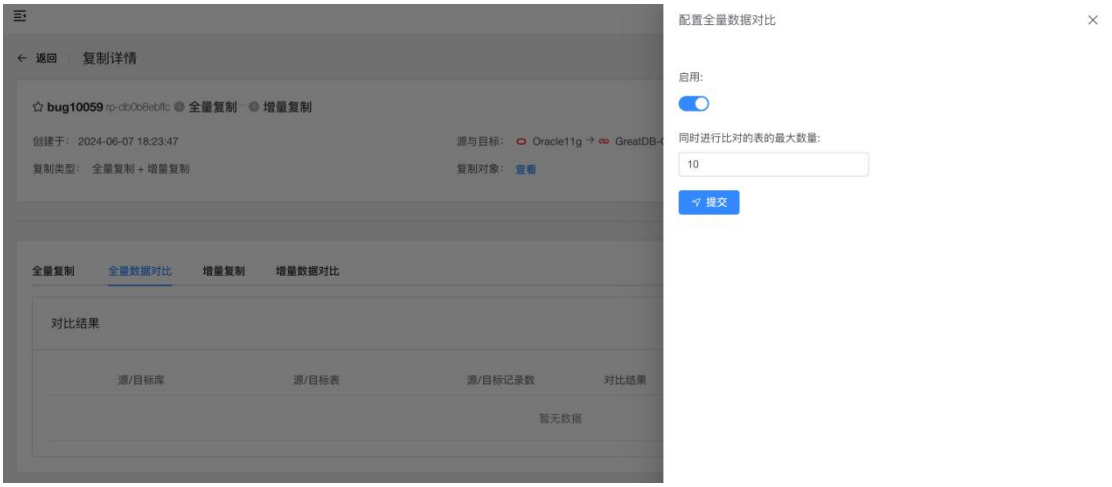
- 按列值拆分：
  - 拆分条件：表预估数据量或表分区平均预估量>20 万
  - 拆分粒度：每 20 万条做一次拆分
  - 拆分策略：当有主键或索引时，自动优先选择离散度高索引拆分，详细的判断流程如下



- 不拆分：对于无主键和索引的表默认的策略是不拆分。

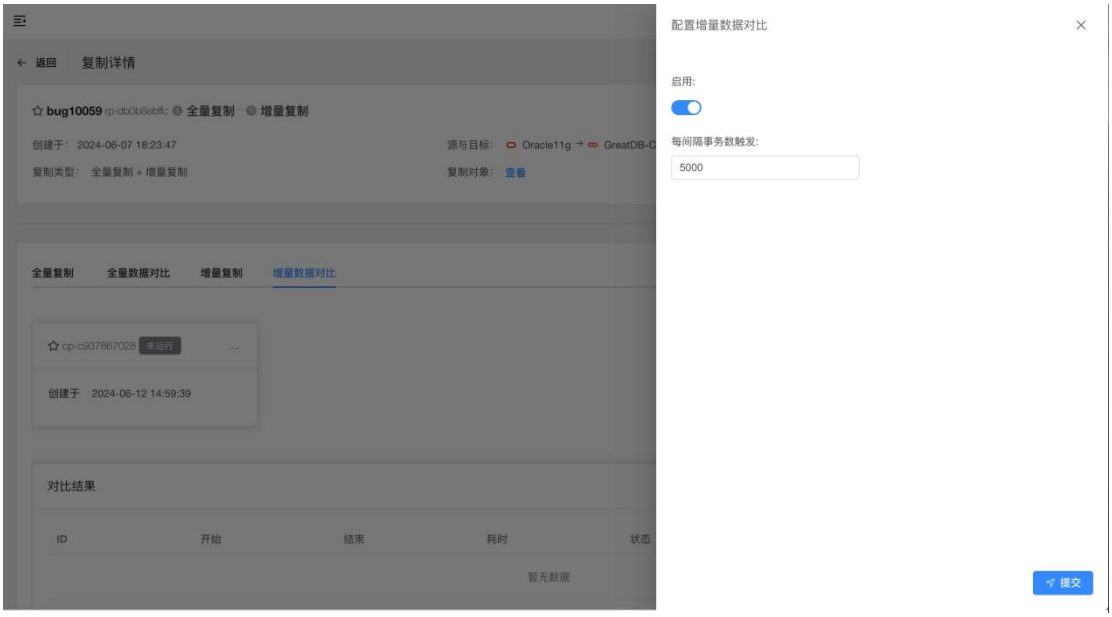
5. 全量数据对比

在复制任务详情页中，用户可以选择在全量复制单张表完成时，启动数据对比



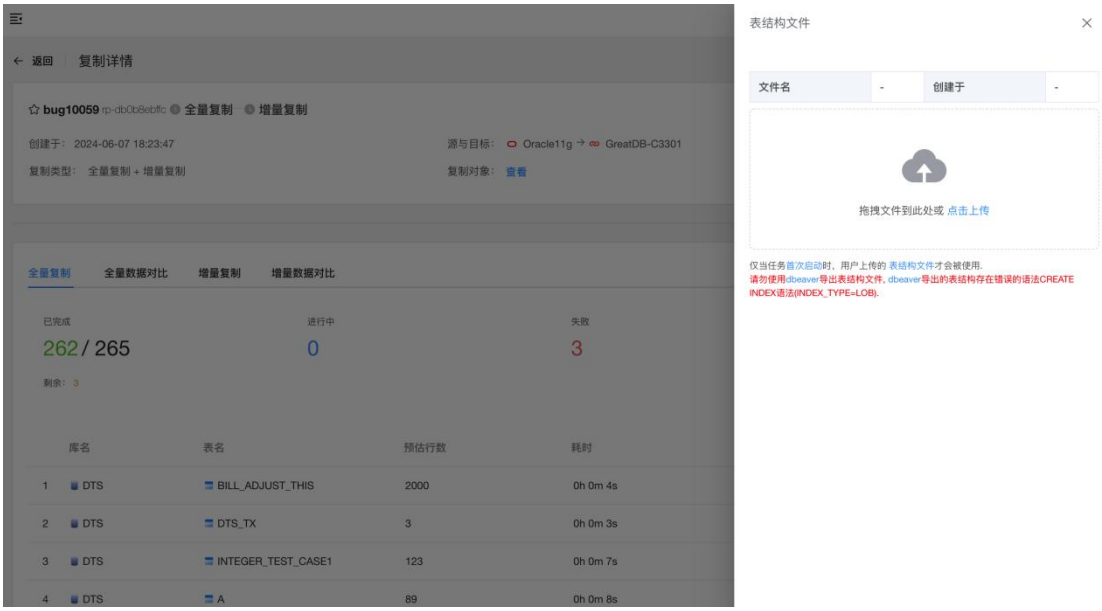
6. 增量数据对比

在复制任务详情页中，用户可以选择在增量复制间隔一定事务数时，执行增量数据的对比。增量数据的对比是累进的，上一次对比的的差异行和下一批次的增量数据合并成对比数据集进行数据对比。在增量数据对比过程中，增量数据的复制并没有停止，所以用户关注最新的一次对比结果即可



7. 启动复制

当前版本 对 Oracle 数据源的初始化，采用 get_ddl 方式获取表结构，get_ddl 随源库表的数量增加时，提取效率下降非常明显，DTS 允许用户【上传表结构文件】来跳过从源库提取表结构 （后续可能移除）

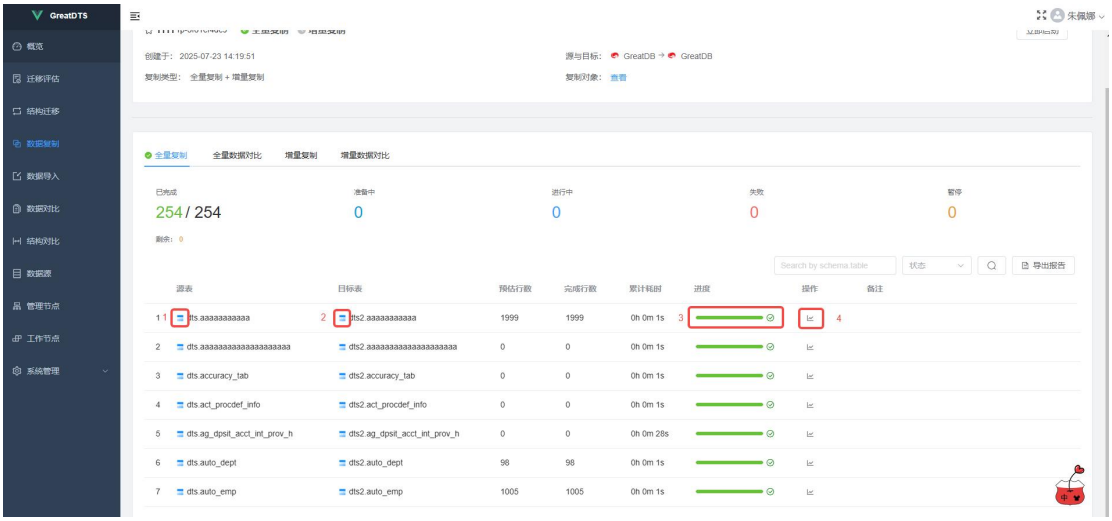


配置调度策略：DTS 允许用户指定任务运行的工作节点

全量任务+增量任务的模式会在全量任务完成后自动启动增量复制

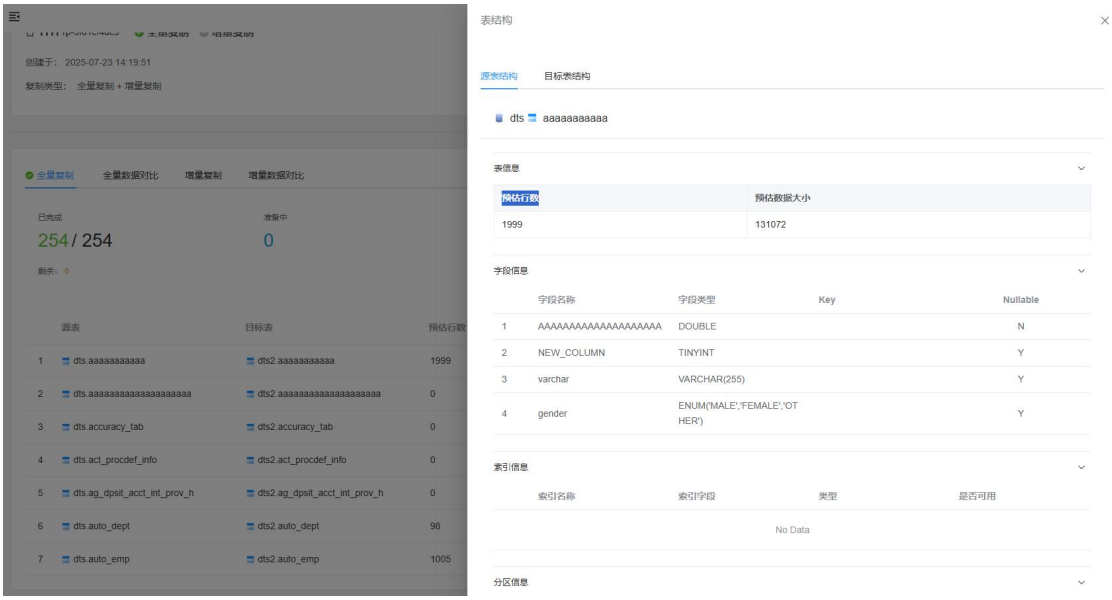
8. 复制详情

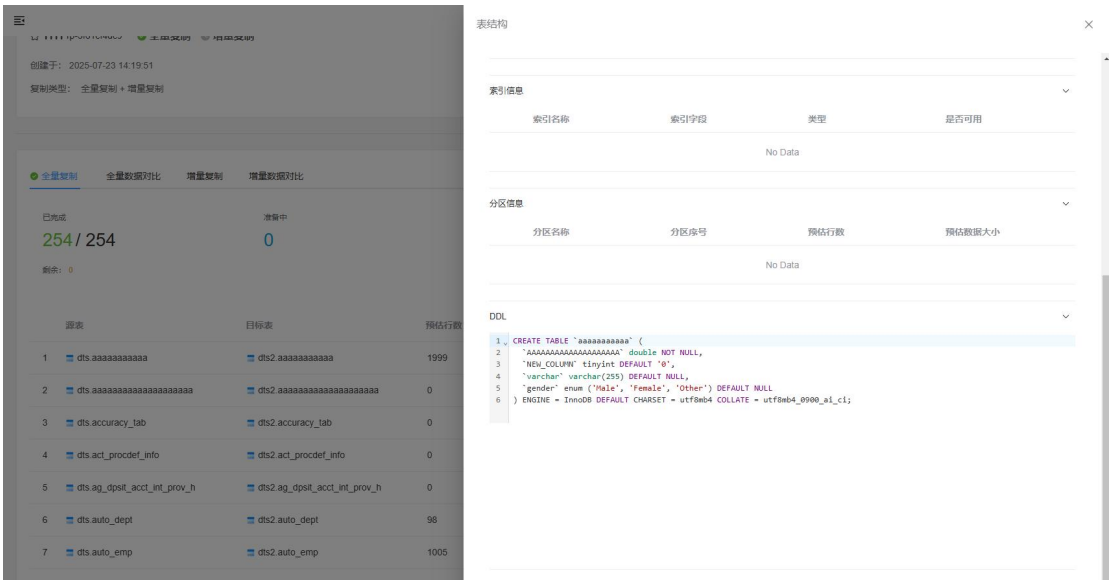
- 全量复制进度详情
- 如下图所示 1、2、3、4 位置都可以点击或悬停查看详细信息。



图中标注的位置 1、2，点击进入可以查看源端和目标端的表的详细信息

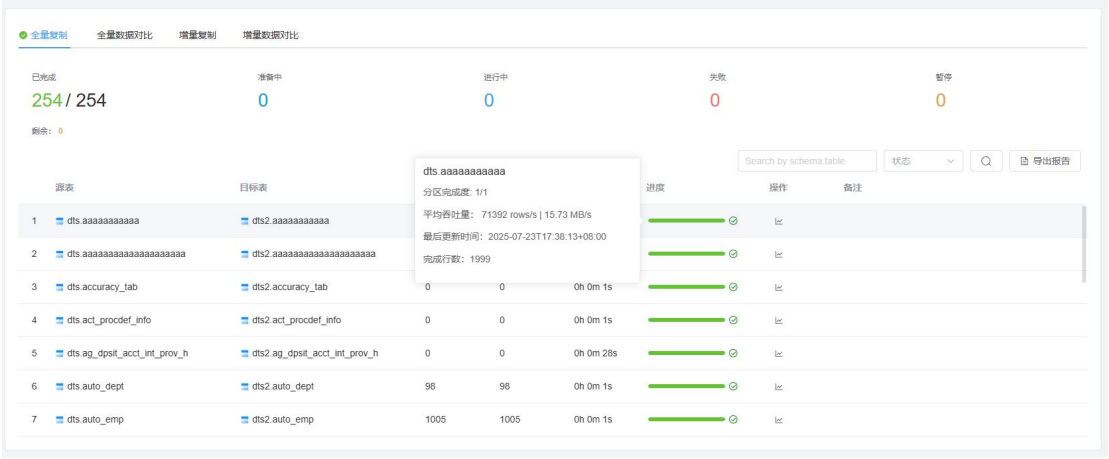
表的预估行数、预估大小、字段信息、索引信息、分区信息、建表 DDL





图中标注的位置 3，鼠标悬停可以实时查看当前表的进度详情

分区完成度（做了拆分，总的分区数量才不为 1）、平均吞吐量、最后更新时间、完成行数。



图中标注的位置 4，点击可以实时查看当前表复制的更多指标

拆分策略、拆分键、预估行数、完成行数、分区完成度、当前进度、累计耗时、平均吞吐量、指标数据保留最大个数，这些指标值为瞬时值，三秒钟产生一个指标数值，最多可以保留 50 个指标数值，如下为更为详细的指标项。

queue.remaining.capacity 缓存队列当前剩余容量（个数）

queue.total.capacity 缓存队列最大容量（个数）

current.queue.size 缓存队列当前已被占用的大小（单位：byte）

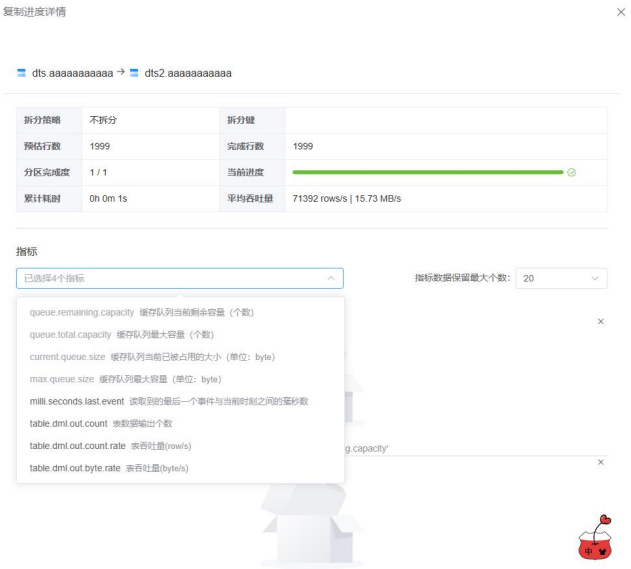
max.queue.size 缓存队列最大容量（单位：byte）

milli.seconds.last.event 读取到的最后一个事件与当前时刻之间的毫秒数

table.dml.out.count 表数据输出个数

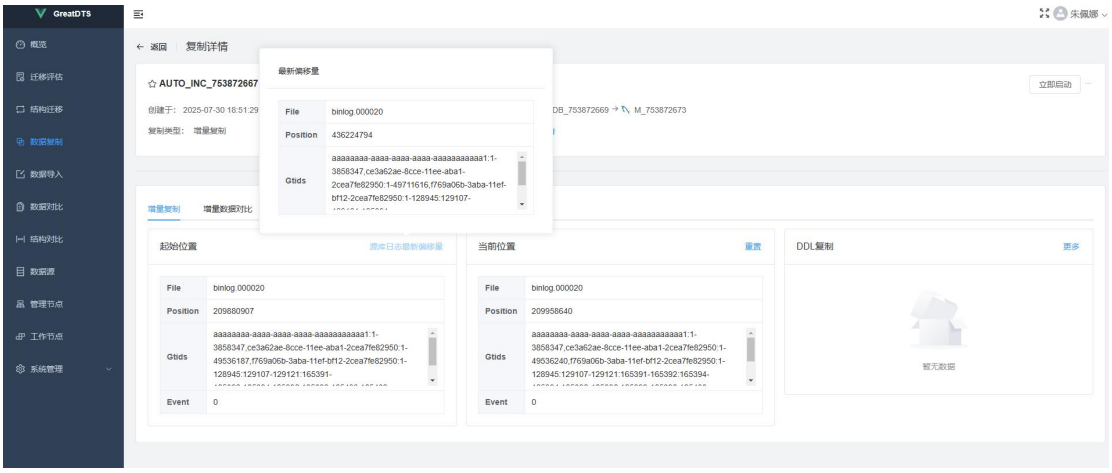
table.dml.out.count.rate 表吞吐量(row/s)

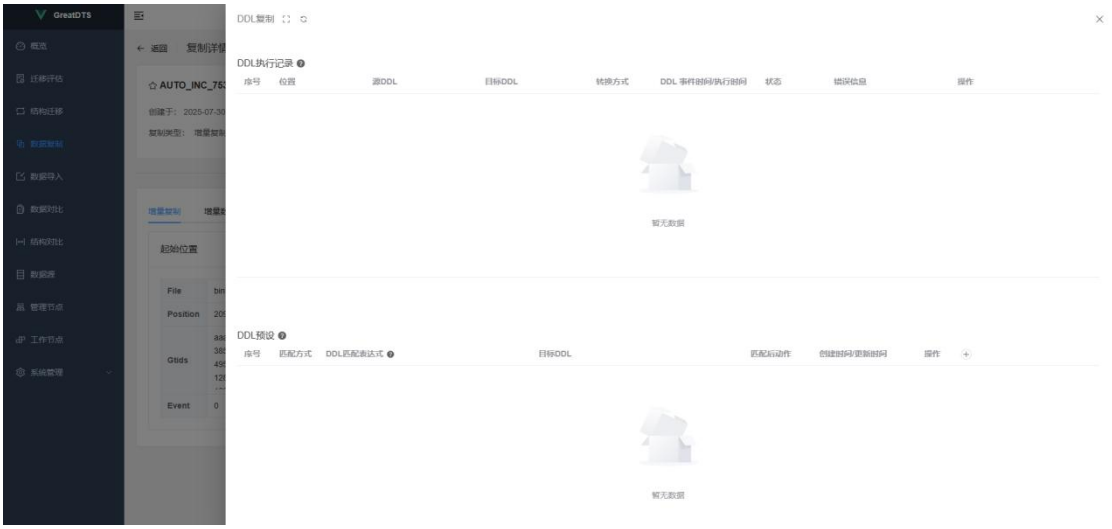
table.dml.out.byte.rate 表吞吐量(byte/s)



- 增量复制查看源端最新的日志位置

620-GA1 新增可以查看增量复制中源端最新的日志位置

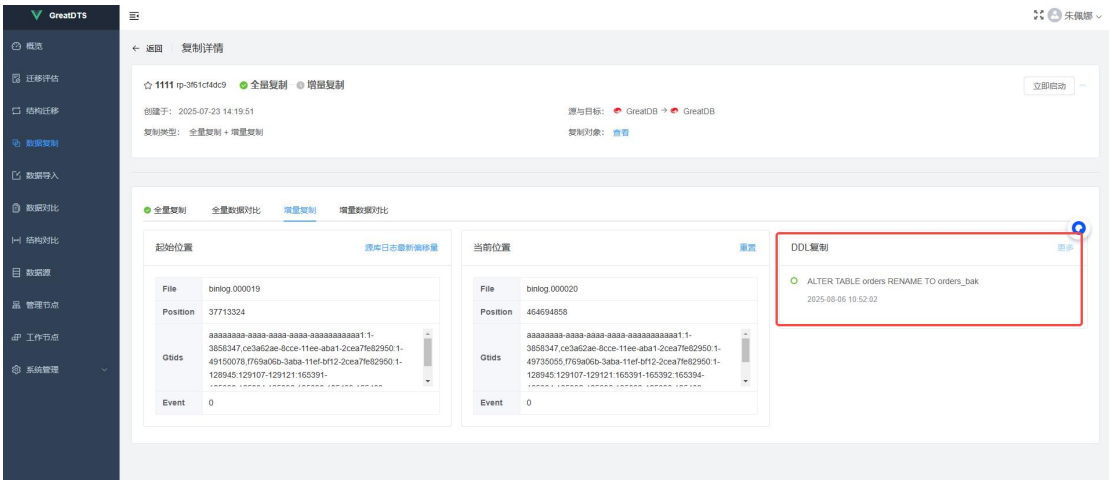


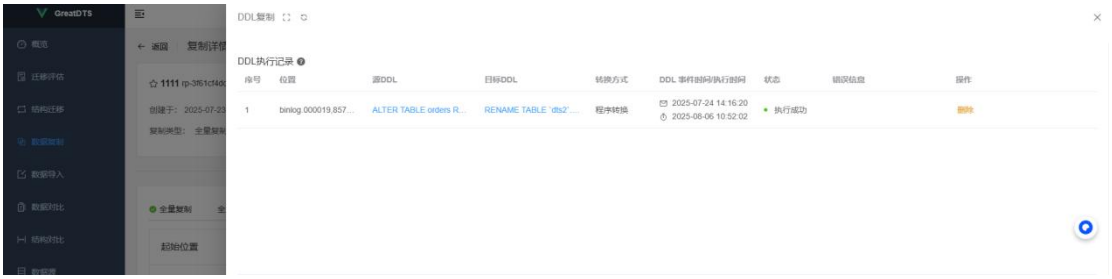


- 增量复制阶段 DDL 复制的配置

620-GA1 增加在增量复制阶段对 DDL 复制做配置可以查看 ddl 的复制详情，增来量复

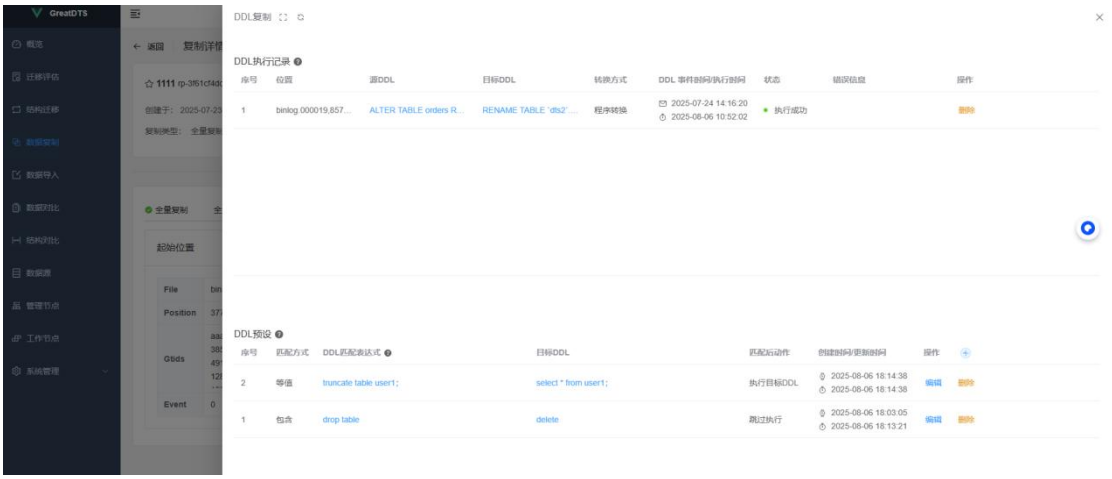
制页面展示的是最新的两条 DDL 信息，点击更多后可以查看所有 DDL 复制的列表。可以才看到 DDL 的转换与执行情况。

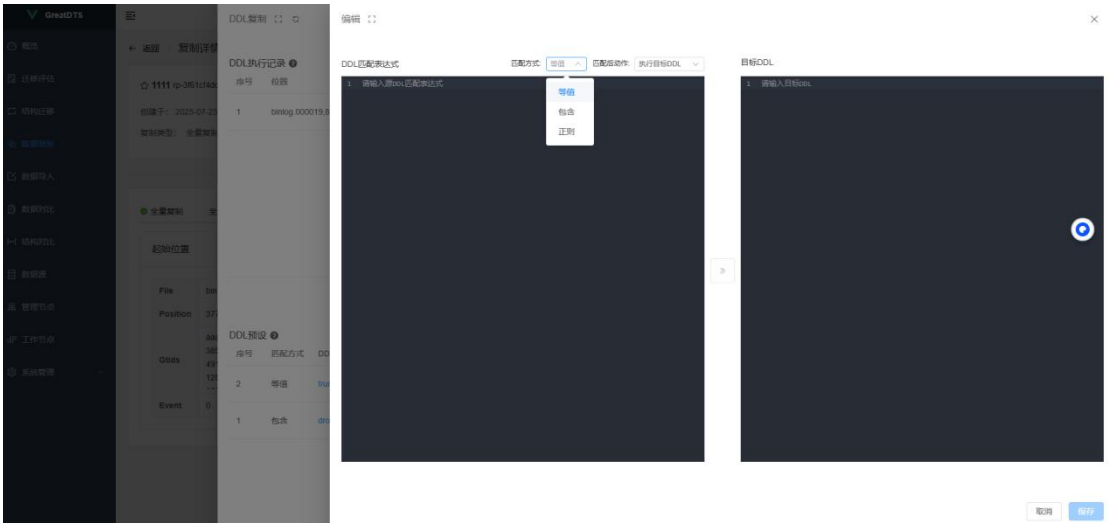


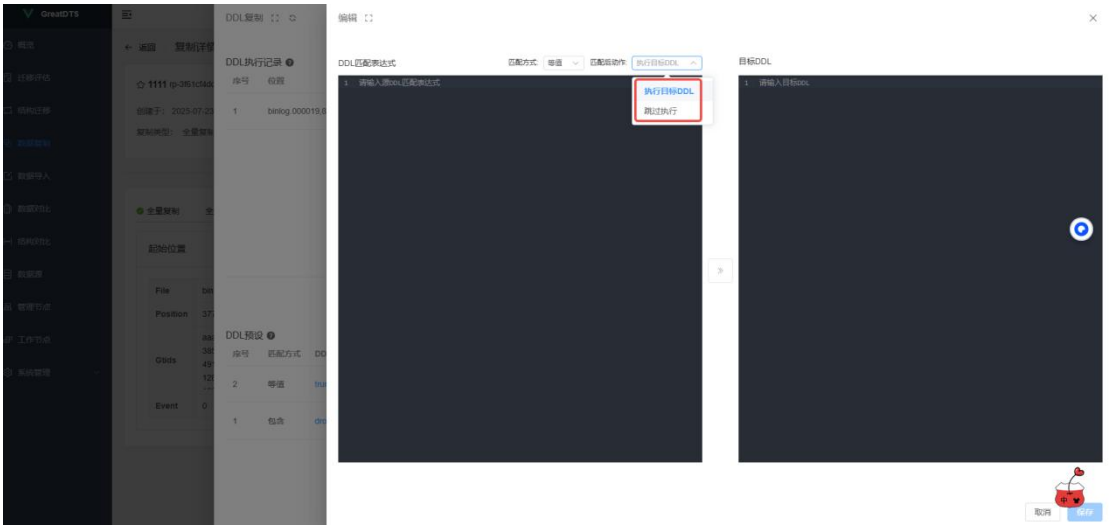


可以对特定 DDL 做预设，例如匹配到源端某些 ddl 可以选择跳过执行或者执行定义好的目标端 DDL

匹配方式有等值、包含和正则；匹配动作有执行目标 DDL 和跳过执行。







9. 任务重置

任务重置会将任务状态重置到次阶段的未运行阶段，如当前运行在全量复制阶段，则重置到未运行的阶段。若是多阶段的任务（全量复制+增量复制）

10. 运行日志

查看和下载任务运行日志，可以从运行历史记录 【操作】一栏 打开日志窗口。日志是对应运行实例的，即每次运行对应一个实例

4.6 数据导入

数据导入是数据增量复制的一种特殊场景。数据导入使用自研组件 Oracle Log Replicator 实现，用于支持从 kafka 导入 Oracle 增量数据，从而实现 oracle 增量数据的复制。如下为 Oracle Log Replicator 组件的详细的配置使用方式

4.6.1 部署 Oracle Log Replicator

Oracle Log Replicator 用于直接解析 Oracle 的 Archive log，将解析后的数据输出到 Kafka。

注意事项：

- 暂不支持的数据类型
  - timestamp with time zone ？
  - timestamp with local time zone?
  - BINARY_FLOAT ？
  - BINARY_DOUBLE ？
- 支持 kafka 认证方式
  - PLAINTEXT
  - SASL/PLAIN

1. OLR 所需权限

   权限文件在：greatsync-oracle/db-scripts/grants.sql

```
SQL
GRANT SELECT, FLASHBACK ON SYS.CCOL$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.CDEF$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.COL$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.DEFERRED_STG$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.ECOL$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.LOB$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.LOBCOMPPART$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.LOBFRAG$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.OBJ$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.TAB$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.TABCOMPART$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.TABPART$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.TABSUBPART$ TO <USER>;
GRANT SELECT, FLASHBACK ON SYS.TS$ TO <USER>;
```

```
GRANT SELECT, FLASHBACK ON SYS.USER$ TO <USER>;
GRANT SELECT ON SYS.IND$ TO <USER>;
GRANT SELECT ON SYS.ICOL$ TO <USER>;
GRANT SELECT ON SYS.V_$ARCHIVED_LOG TO <USER>;
GRANT SELECT ON SYS.V_$DATABASE TO <USER>;
GRANT SELECT ON SYS.V_$DATABASE_INCARNATION TO <USER>;
GRANT SELECT ON SYS.V_$LOG TO <USER>;
GRANT SELECT ON SYS.V_$LOGFILE TO <USER>;
GRANT SELECT ON SYS.V_$PARAMETER TO <USER>;
GRANT SELECT ON SYS.V_$STANDBY_LOG TO <USER>;
GRANT SELECT ON SYS.V_$TRANSPORTABLE_PLATFORM TO <USER>;
```

2. 解压 greatsync-oracle-6.0.3-RC-1-7bcf9c96.tar.gz

```
Shell
greatsync-oracle
./
├── GREATSYNC_ORACLE-VERSION
├── USAGE.md
├── bin
│   ├── boot.sh
│   ├── greatsync-oracle
│   └── tabls2json.sh
├── db-scripts
│   ├── gencfg.sql
│   └── grants.sql
└── task-example
    ├── start.sh
    ├── status.sh
    ├── stop.sh
    └── sync.json
```

- grants.sql ： 应用所需要的 ORACLE 权限
- USAGE.md: 使用文档

3. 创建 WorkSpace

task-exapmle 是工作目录的示例，拷贝一个新的目录，如 task-1 （和 greatsync-oracle 同级目录）

```
Shell
# tree ./ -L 2
```

```
./
├── greatsync-oracle
│   ├── bin
.......
│   ├── scripts
│   └── task-example
└── task-1
    ├── start.sh
    ├── status.sh
    ├── stop.sh
    └── sync.json
```

```
Shell
# 进入 task-1 目录，配置任务
 > cd tsak-1
 > cat sync.json
 {
  "version": "1.3.0",
  "dump-redo-log": 0,
  "dump-raw-data": 0,
  "dump-path": "var/dump",
  "log-level": 4,
  "trace": 0, // TRACE 信息，调试时用 98303(观察 sql 执行)，131071（最详尽），
  "write-pid": 1,
  "source": [
    {
      "alias": "S1",
      "name": "DB",
      "reader": {
        "type": "online",  // online(在线模式,需要连接示例)，offline（离线模式，需要提前导出字典）
        "user": "TESTUSR1",
        "password": "TESTUSR1PWD",
        "server": "//localhost:11522/xe",
        "start-scn": 357199, // 同步开始的 scn
        "start-seq": 6,      // start-scn 对应的 seq，加速定位，可选参数
```

```
        "dict-with-scn": 0,
        "use-obj-filter": 0, // 只采集符合 filter->table 规则的表字典信息，开启后在几万个表的 db 同步少量表横明显提速
        "arch-read-sleep-us": 10000000, // 归档重试间隔
        "arch-read-tries": 10, // 归档文件的重试次数 范围 1~1000000000，总重试时间 archReadSleepUs * arch-read-tries
        "path-mapping": [   //归档路径映射，两个一组，  归档路径 1，主机路径 1，归档路径 2，主机路径 2
                          "/u01/app/oracle/archived", //归档文件所在目录的上一层或以上

"/home/wooyea/work/wooyea/oracle_redo/greatsync-oracle-test/11-xe/var/archived"
              ]
      },
      "arch": "online", // 归档获取模式(离线模式无效) path(本地路径), online(在线短连接，读取完字典关闭), online-keep(在线长连接)
      "load-table-key-online": 1, // 从最新字典表加载 pk/uk,默认开启，会保持长连接
      "flags": 53, //  online 53, offline 37
      "format": {
        "schema": 7,
        "type": "protobuf", // 输出格式，protobuf（dts kafka），json（本地文件输出）
        "interval-dts": 8,
        "interval-ytm": 4,
        "column": 2,
        "db": 3,
        "rid": 1,
        "scn-all": 3,
        "scn": 0,
        "timestamp-all": 1
      },
      "memory-min-mb": 128,
      "memory-max-mb": 4096, //生产建议 4096+
      "filter": {
        "table": [
          // {“table”: “owner2.table2”, “key”: “col1, col2, col3”}  owner/table 可用正则表达式，key 可选，用于指定逻辑键
          {"owner": "TESTUSR1", "table": ".*"}
```

```
        ]
      },
      "state": {   // 同步进度存储
        "type": "disk",
        "path": "var/checkpoint",
        "interval-s": 600,
        "interval-mb": 100,
        "keep-checkpoints": 100,
        "schema-force-interval": 20
      }
    }
  ],
  "target": [
    {
      "alias": "K1",
      "source": "S1",
      "writer": {
        // 输出到 dts kafka, source.format.type: "protobuf"
        "type": "kafka",
              "topic": "topica", // 默认的 topic
        "max-message-mb": 500,
        "properties": {
          "bootstrap.servers": "127.0.0.1:9092",
          "enable.idempotence": "true",
          "queue.buffering.max.messages": "200000"
        },
        "poll-interval-us": 100000,
        "queue-size": 6553,
        "table-groups":{ //分组路由配置, 可选
          "TOPIC_T1": "TESTUSR1.T1", // 可以配置多个对象, 对象的格式为: 表模式.表名。对象之间用逗号分割, 不支持正则表达式, 对象名称严格匹配, 字母忽略大小写。
          "TOPIC_T234": "TESTUSR1.T2,TESTUSR1.T3"
        }
      }
      /*
      "writer": {
        // 输出到本地文件, source.format.type: “json”
        "type": "file",
        "output": "var/output/output.json",
```

```
        "format": "%F-%T",
        "new-line": 1,
        "max-file-size": 1073741824
      }
      */
    }
  ]
}
```

4. 任务配置

Oracle Log Replicator 可以工作在“在线模式”和离线模式。

在线模式：

```
Shell
vi sync.json

"source": [
    {
      "alias": "S1",
      "name": "DB",
      "reader": {
        "type": "online",       // online
        "user": "TESTUSR1",
        "password": "TESTUSR1PWD",
        "server": "//localhost:11522/xe",
        # start-scn  同步的开始的 scn (V$ARCHIVED_LOG FIRST_CHANGE#)
        "start-scn": 3688639,
        # start-seq 归档文件的序列 ID （v$archived_log RECID）
        "start-seq": 20,
        "path-mapping": [
            "/u01/app/oracle/archived", //归档文件所在目录的上一层或
以上
            "/home/wooyea/work/wooyea/oracle_redo/greatsync-oracle-
test/11-xe/var/archived"
              ]
      },
      ...............
```

参数说明：

参数	说明
user	用户名
password	用户密码
server	oracle 连接 url
start-scn	增量数据的起始位置。一般从 V$ARCHIVED_LOG 表中的 FIRST_CHANGE#
start-seq	归档文件的 RECID。V$ARCHIVED_LOG 表的 RECID
path-mapping	V$ARCHIVED_LOG 表中 NAME 的目录和本地目录的映射。两个为一组。 一般 Replicator 运行的服务器不是 ORACLE Server 所在的服务器，归档日志被拷贝到 Replicator 所在的服务器，这时需要配置日志目录的映射。

离线模式:

离线模式 是直接利用在线模式生成的字典（var/checkpoint），而不需要每次都连接 Oracle Server 去获取。离线模式不允许表发生 DDL

```
Bash
拿到的 DB-chkpt-82320989686.json
放到 var/checkpoint 里，改名为 DB-chkpt-${start-scn}.json
编辑 DB-chkpt-scn1.json 里前面大约二十行
把 scn,schema-scn 改${start-scn},
更新改 restlog  activation 改， 如果第一行有 minitran 去掉
offset 改 0
重点是 log-archive-format， 要根据拿到的归档改成对应的格式
```

```
Shell

> ls var/checkpoint/
# 将第一个文件名变更为 DB-chkt-${start-scn}.json，删除其他文件
> mv DB-chkt-${最小的 scn}.json DB-chkt-${start-scn}.json
# 编辑 DB-chkt-${start-scn}.json，将一下内容变更为${start-scn}
> vi DB-chkt-${start-scn}.json
```

```
{"database":"DB","scn":${start-
scn},"resetlogs":1092497716,"activation":2217039615,"time":116833647
5,"seq":${startr-seq},"offset":0,"big-endian":0,"context":"","con-
id":0,"con-name":"ncsmpdb","db-recovery-file-dest":"","db-block-
checksum":"TYPICAL","log-archive-dest":"","log-archive-
format":"thread_%t_seq_%s.%h.%a","nls-character-
set":"ZHS16GBK","nls-nchar-character-set":"AL16UTF16","supp-log-db-
primary":0,"supp-log-db-all":1,
"online-redo":[],
"incarnations":[],
"users":[
"ACCOUNTING",
"BILLING",
"SETTLE"],
"schema-scn":${start-scn},
```

参数	说明	
start-scn	需要同步的第一个归档文件的 start scn	
format	归档日志的文件格式，使用一下 SQL 查询获得： SELECT NAME, VALUE FROM V$PARAMETER WHERE NAME LIKE 'log_archive_format';	

```
JSON
> vi sync.json

"source": [
    {
```

```
      "alias": "S1",
      "name": "DB",
      "reader": {
        "type": "offline",
        "start-scn": 3688639,
        "start-seq": 20,
      },
      ...............
```

```
Shell
#移除 DB-chk.json 进度文件
> rm db-chk.json
```

配置需要捕获的表：

```
Shell
"source": [
    {
      "filter": {
        "table": [
          // {“table”: “owner2.table2”, “key”: “col1, col2, col3”}  owner/table 可用正则表达式，key 可选，用于指定逻辑键
          {"owner": "TESTUSR1", "table": ".*"}
        ]
      },
    }
  ],
```

配置输出到 kafka

```
Shell
 > vi sync.json

 "source": [
    {
      "format": {
        "type": "protobuf",
      },
```

```
    }
  ],

 "target": [
    {
      "alias": "K1",
      "source": "S1",
      "writer": {
        // 输出到 dts kafka, source.format.type: "protobuf"
        "type": "kafka",
        "topic": "topic",
        "properties": {
          #sasl 配置, 没有则移除
          "security.protocol": "sasl_plaintext",
          "sasl.mechanism": "PLAIN",
          "sasl.username": "client",
          "sasl.password": "client-secret",
          "bootstrap.servers": "127.0.0.1:9092"
        }
      }
    }
  ]
```

其他的参数说明

参数	可选	说明

table-groups	是	分组路由配置。
		"table-groups":{
		"TOPIC_T1": "TESTUSR1.T1", // 可以配置多个对象（1 到 N），对象的格式为：表模式.表名。对象之间用逗号分割，不支持正则表达式，对象名称严格匹配，字母忽略大小写。
		"TOPIC_T234": "TESTUSR1.T2,TESTUSR1.T3"
		}
		都没有匹配成功，走到默认的 topic。

5. 启动和停止

```
Shell
```

```
# 启动
.task-1/start.sh
# 停止
.task-1/stop.sh
```

4.6.2 Oracle RAC 版本使用说明

1）配置文件

olr 多路归档排序输出，配置参考：rac-merge.json

样例如下：

```
JSON
{
    "rac-node-1": {
        "path": "/home/tangjie/rac-merge-1.json"
    },
    "rac-node-2": {
        "path": "/home/tangjie/rac-merge-2.json"
    }
}
```

a. 每个节点的配置必须以 rac-node-{n} 命名，其中 n 必须是以 1 为起始的有序数字。

b. path 取值为单节点 olr 配置文件 sync.json 存放路径，内容参考单节点 olr 的 sync.json 的配置内容。

rac-node-{n}.path 指定的配置文件中需要注意的是：

- source.state.path 取值必须不同，并且对应路径下要存放它们各自的字典文件（DB-chkpt-XXXXX.json）。
- need-rac-scn 必须配置 1，scn-all 必须配置 3。
- writer 双节点的输出类型要保持一致，即双节点同时输出到文件或者双节点同时输出到 kafka，它们的配置要保持一致。

2）使用说明

a. 启动脚本：start-rac.sh

b. 跟踪 grep "probe rac schedule" stdout.txt 的日志，如果出现类似的提示：

```
Plain Text
2024-10-24 14:02:38 taskRecycleThread SLIENT 00000 probe rac schedule: RAC, output rows: 4985, handle the redo files transaction merge stop.
2024-10-24 14:02:42 taskRecycleThread SLIENT 00000 probe rac schedule: RAC, output rows: 4985, handle the redo files transaction merge stop.
```

说明 olr 已经无法合并。

c. 执行：stop.sh

继续跟踪 grep "probe rac schedule" stdout.txt 的日志，出现

```
Plain Text
2024-10-24 14:02:53 taskRecycleThread SLIENT 00000 probe rac schedule: RAC, output rows: 5065, handle the redo files transaction merge tasks execution completed.
```

说明 olr 已经将无法继续合并的数据直接刷新输出，并退出。

至此，合并流程结束。

3) 注意事项

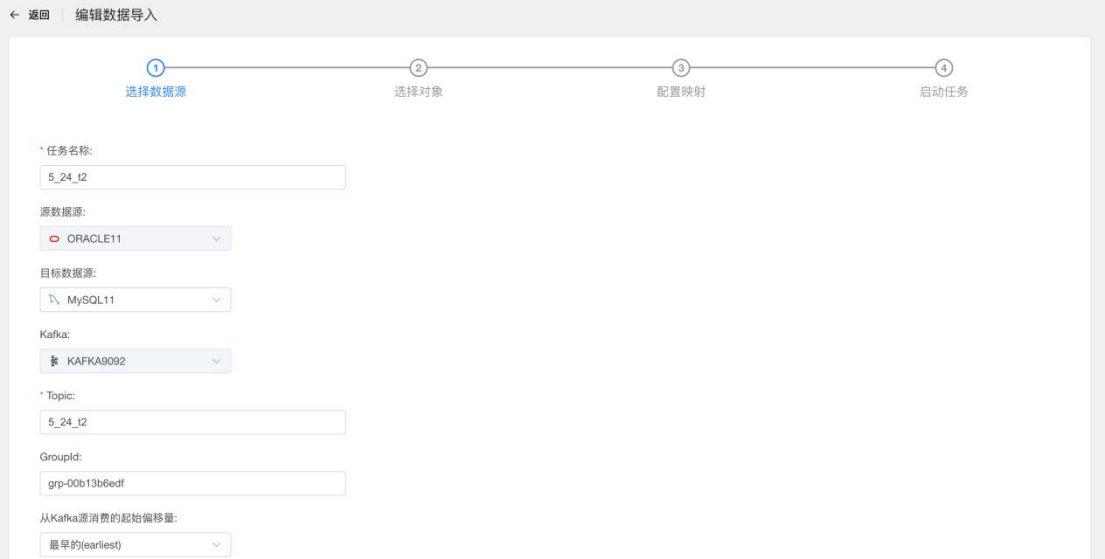
a. 输出 merge stop 日志意味着无法继续合并，也就是说在输出 merge stop 日志前，程序处于自主解析合并输出状态，此时，如果执行 stop.sh，则会立刻停止解析并退出程序。此时，程序会保存断点，后续可以执行启动脚本 start-rac.sh 从断点处恢复解析。

b. 输出 merge stop 日志后，如果不希望将无法继续合并的数据直接刷新输出，则可以执行 stop.sh force 强制合并结束。

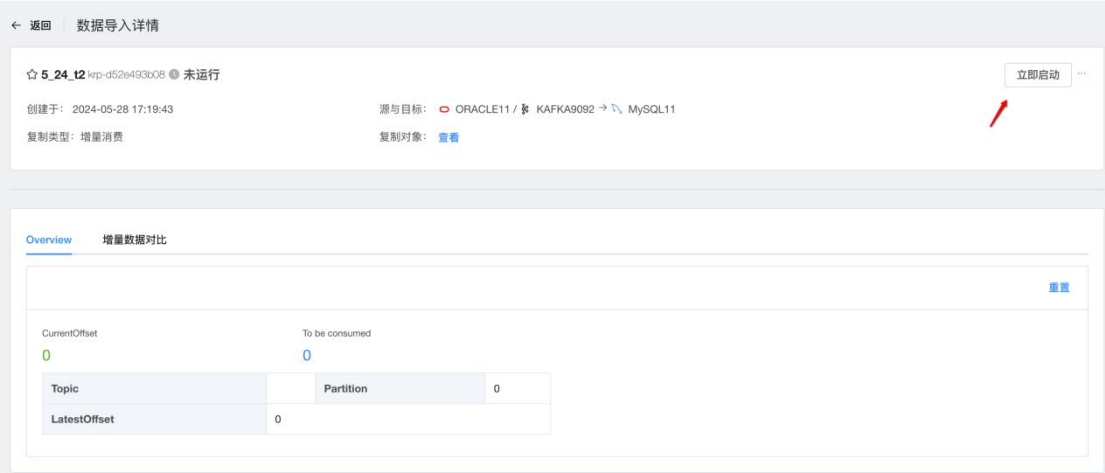
4.6.3 配置数据导入

1. 创建 KAFKA 数据源和 ORACLE 数据源 （略，请参考 【创建数据源】）
2. 创建任务

选择源数据源、目标数据源。配置 kafka 的 topic，配置 kafka 消费的起始偏移



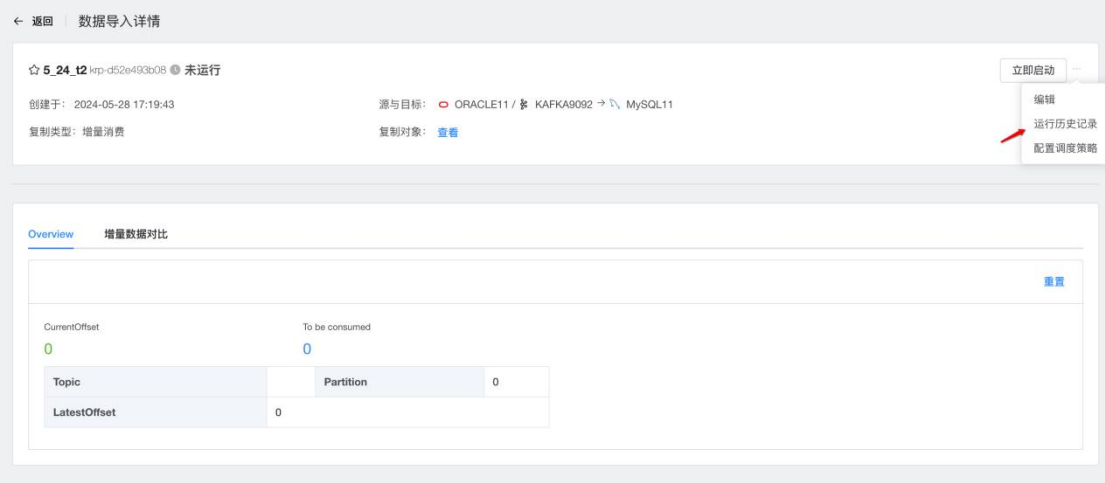
4.6.4 启动任务



页面行可以查看当前的 kafka 数据消费情况

4.6.5 运行日志

从【运行历史记录】中，可以查看和下载任务的运行日志



4.6.6 Kafka topic 数据排查

[check-1.0-SNAPSHOT-jar-with-dependencies.jar]

使用步骤

1. 将上面的包放入 Kafka 的 libs 目录下
2. 查找并输出某个数据

比如想查找 CENTRE 库下的 MEMBER_OPEN_LOG 表且字段名称为 MOBILE_PHONE 值为 13030606701 的数据，假设 topic 的名称是 topic_hongyuan0324，该 topic 最大的 offset 是 1000000 结果输出到当前目录下文件 CENTER_MEMBER_OPEN_LOG_13030606701.log，在 offset 为 999999 时自动结束脚本。

```
Shell
nohup ./bin/kafka-console-consumer.sh --bootstrap-server
192.168.105.95:9092 --topic topic_hongyuan0324 --partition 0 --from-
beginning --consumer.config config/config.properties --formatter
"com.greatdb.kafka.formatter.ProtobufResponseMessageFormatter" --property
schema=CENTRE --property table=MEMBER_OPEN_LOG --property
col_name=MOBILE_PHONE --property col_value=13030606701 --property
auto.exit.offset=999999 >> CENTER.log 2>&1 &
```

要注意的是：

- 上面的命令是在 Kafka 的根目录下执行的
- --property auto.exit.offset=999999 中的 999999 一般是该 topic 最大的 offset 减去 1，否则可能出现无法自动停止的情况
- --consumer.config config/config.properties 可能是 Kafka 的权限相关的，根据实际情况进行配置和不配置

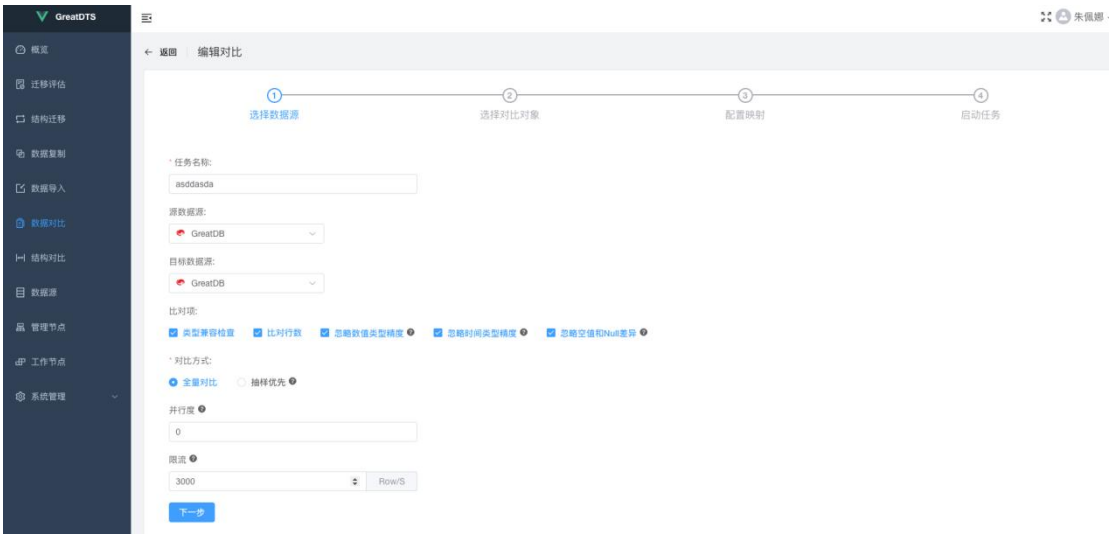
参数说明：

- `--topic <topic_name>` 指定要读取的 topic 名称（必选配置）
- `--partition 0` 指定 partition 为 0（必选配置）
- `--from-beginning` 从头开始读（必选配置）
- `--formatter "com.greatdb.kafka.formatter.ProtobufResponseMessageFormatter"` 指定输出为 DTS 里定制化的 protobuf 格式的数据（必选配置）
- `--property schema=<schema_name>` 过滤出某个 schema:`<schema_name>` 的数据（可选配置）
- `--property table=<table_name>` 过滤出某个表：`<table_name>` 的数据（可选配置）
- `--property col_name=<列名>` 和 `--property col_value=<列的值>` 必须配合一起使用，用于过滤出某个列名它的等于配置的列的值的数据（可选配置）
- `--property auto.exit.offset=<自动停止的 offset>` 配置后可以自动到达该 offset 后停止，一般指定为 topic 的最大 offset 减去 1 后的值（可选配置）

4.7 数据对比

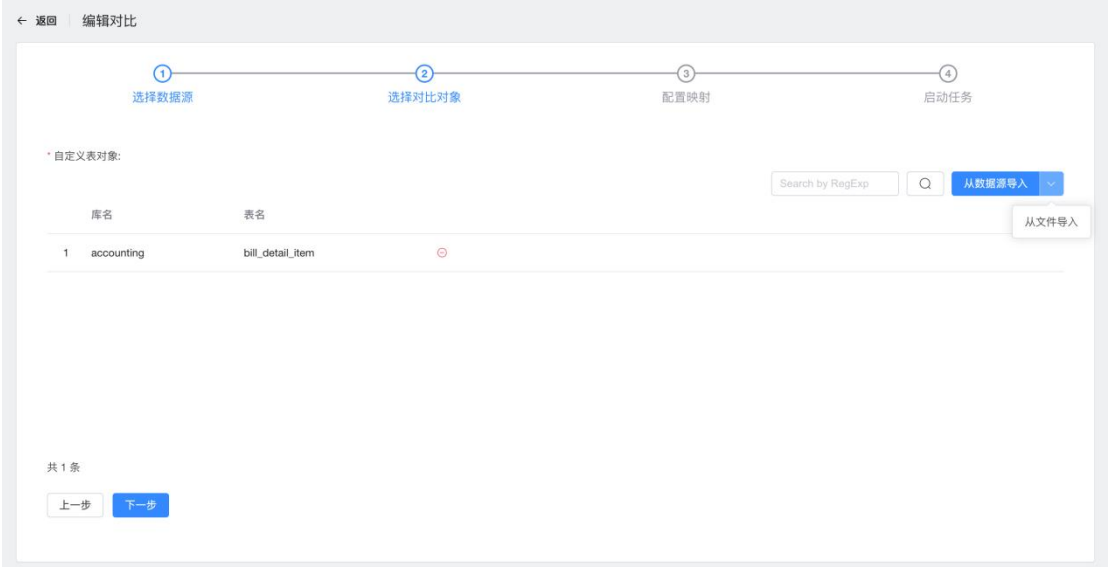
GreatDTS 数据对比支持多种同构、异构数据源之间的结构对比、数据对比。对比结果输出差异报告和修复 SQL。

4.7.1 创建数据对比

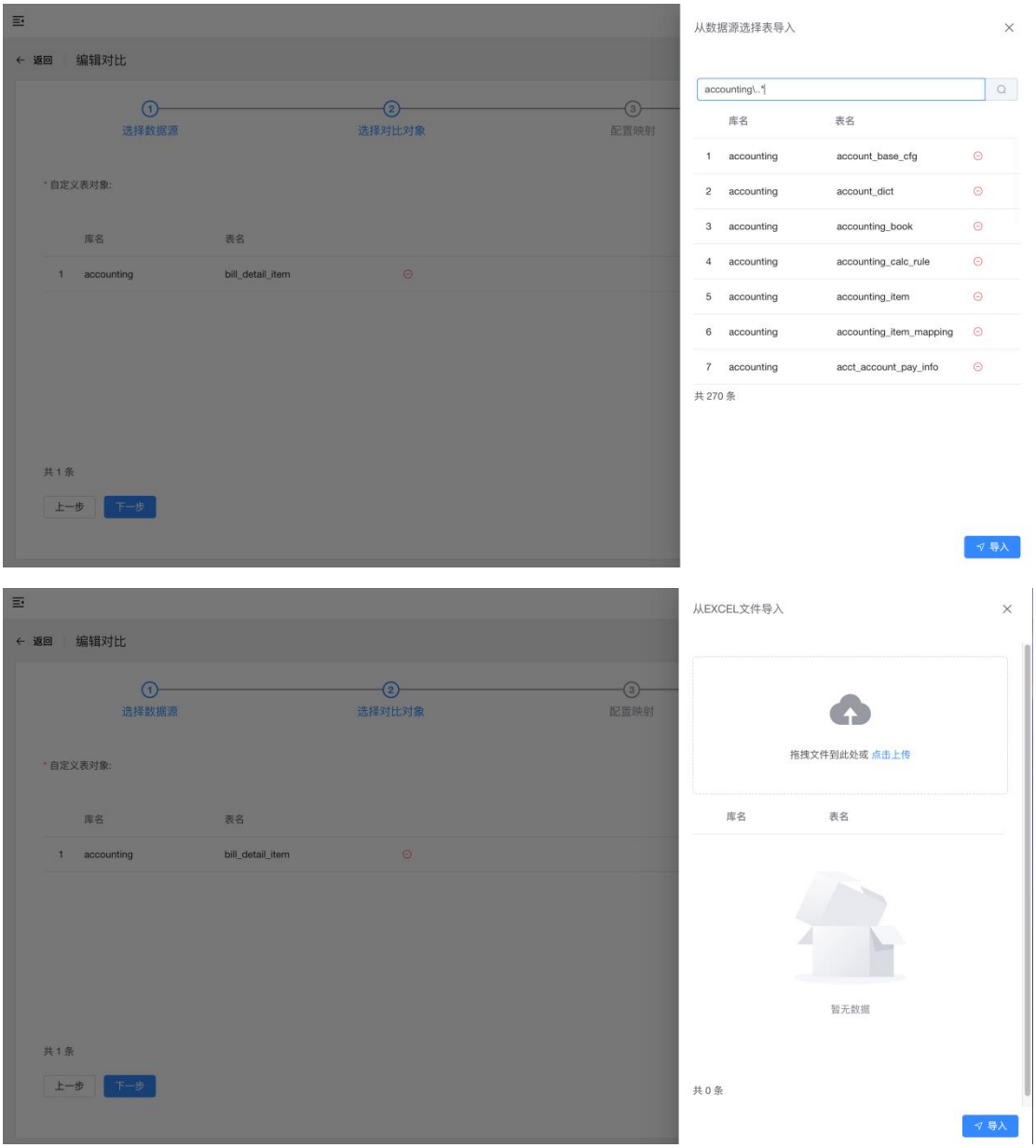


参数	说明

源数据源	源数据源
目标数据源	目标数据源
对比项	结构对比 数据对比：对比行数（表总行数）、忽略数据类型精度差异（NUMBER 类型的精度差异，取最小的一端）、忽略时间类型精度差异（取最小的一端）
并行度	数据对比的并行度。默认为 0，会自动匹配 CPU 的可用核心数。即核心数为多少并行度就为多少。
限流	限制每秒从源和目标数据源读取的行数。默认 0 不限制。当设置限流时，并行度无论是多少都将被设置为 1。即有限流就没有并行度



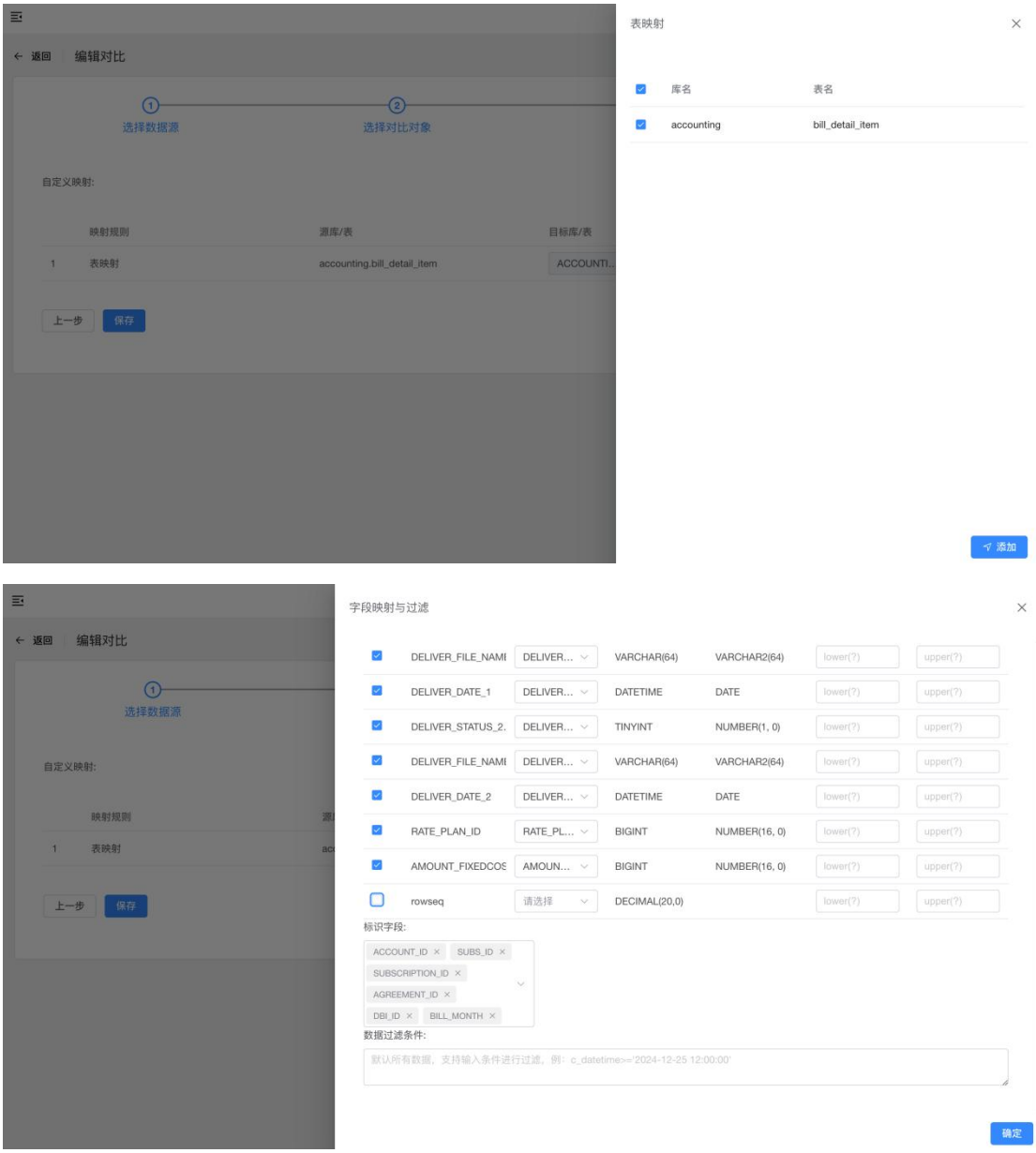
选择对比对象：对比对象可以从源数据源导入（正则匹配）或者从 excel 文件导入。



配置映射：

1. 当源数据库和目标数据库存在表名称、列名称不同时，可以指定映射规则，
2. 指定表的列范围和对比的数据集范围
3. 指定标示字段，默认情况下 DTS 会自动识别表主键或唯一键作为标示字段，用户也可直接指定
4. 不配置映射规则，默认表名、列名完成相同、数据集范围为全表

首先，从右上角按钮【添加表映射】点击，在弹出的列表中勾选需要添加的规则，添加



4.7.2 执行数据对比

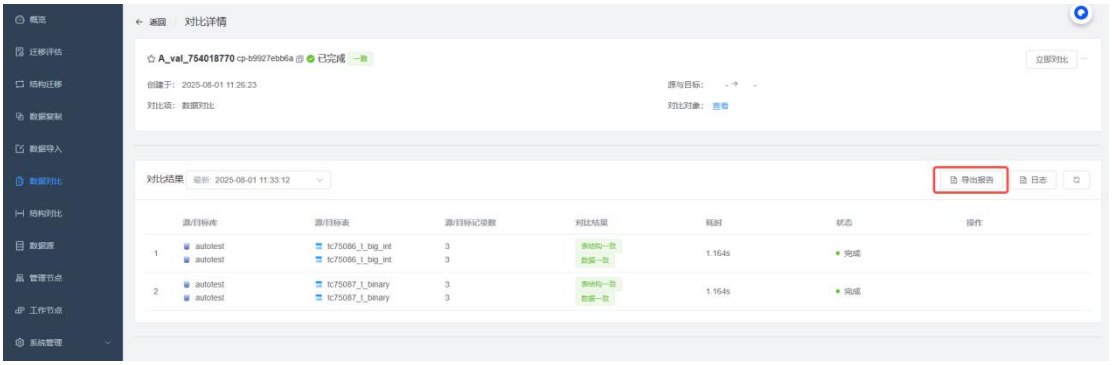
指定数据对比任务的入库可以是列表页面也可以在任务的详情页。

执行完成后，显示结果信息

数据对比任务可以多次执行，可以通过下拉框选择查看任意一次的执行结果报告。报告表格中，操作一览，有四个操作：【查看差异的详情】、【下载差异文件】、【下载修复 SQL】、和【执行修复 SQL】。

当执行任务发生异常时，可以通过对比结果页【日志】或【运行历史记录】中查看和下载日志

支持导出数据比对的报告



4.8 结构比对

创建结构比对

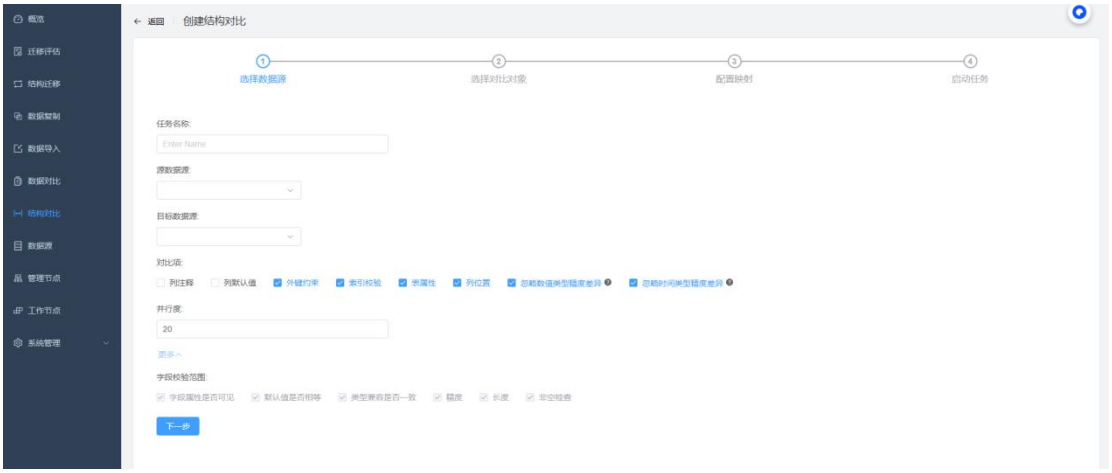
- 比对项

目前支持的比对项：列注释、列默认值、外键约束、索引校验、表属性、列位置、忽略数值类型精度差异（当源数据源与目标数据源存在精度差异时，忽略精度差异部

分）、忽略时间类型精度差异（当源数据源与目标数据源存在精度差异时，忽略精度差异部分）

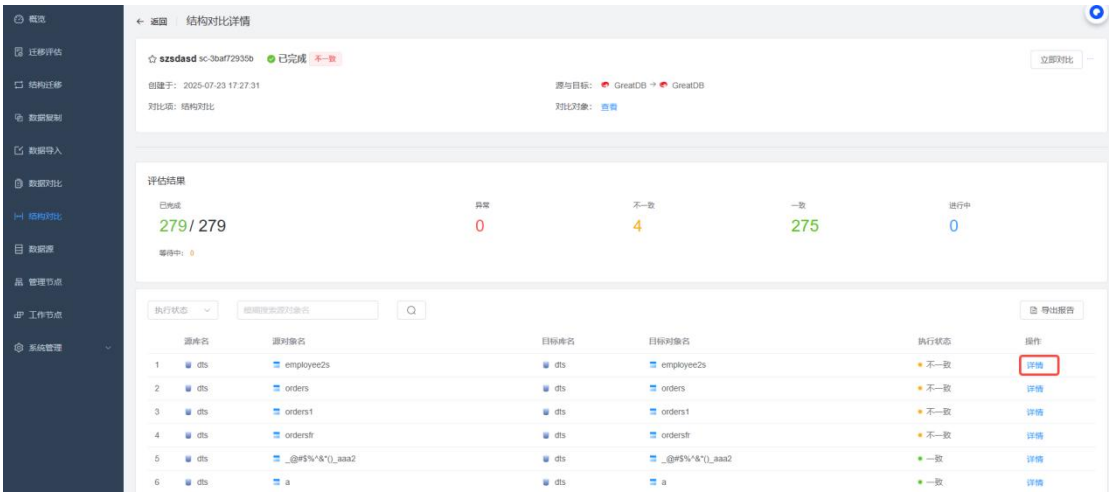
字段校验范围

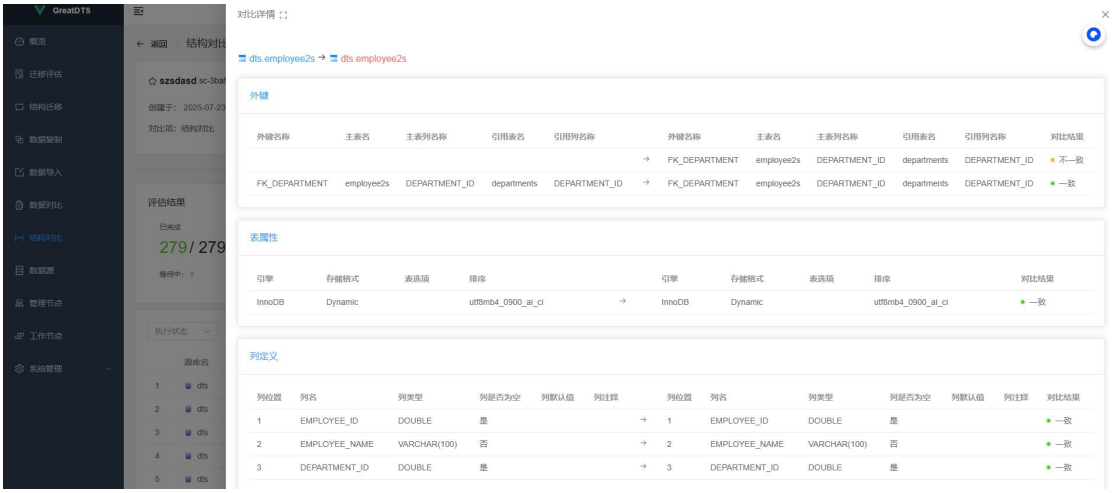
支持的字段校验范围：字段属性是否可见、默认值是否相等、类型兼容是否一致、精度、长度、非空检查



比对详情

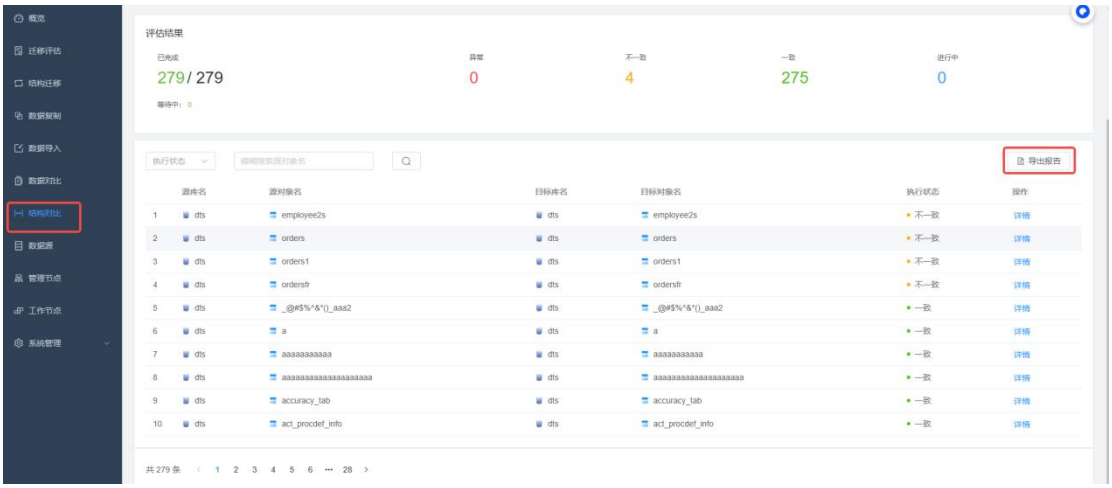
- 查看每张表的比对结果





- 导出报告

支持导出比对结果的报告

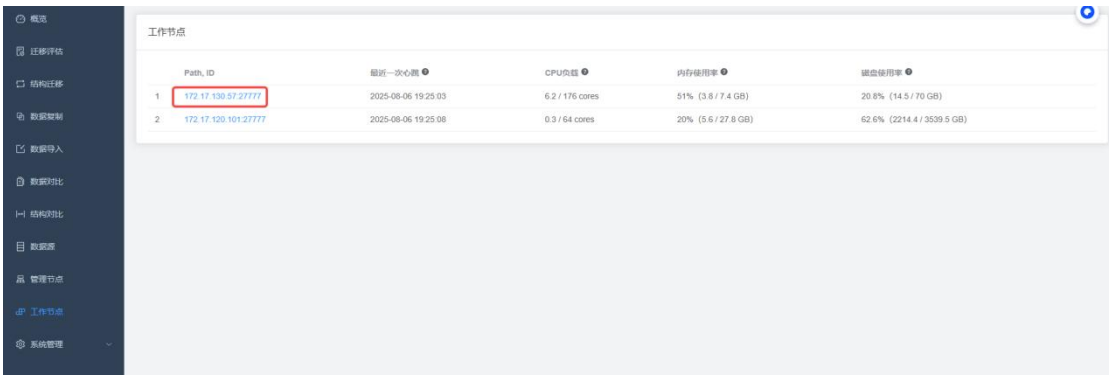


4.9 管理节点和工作节点

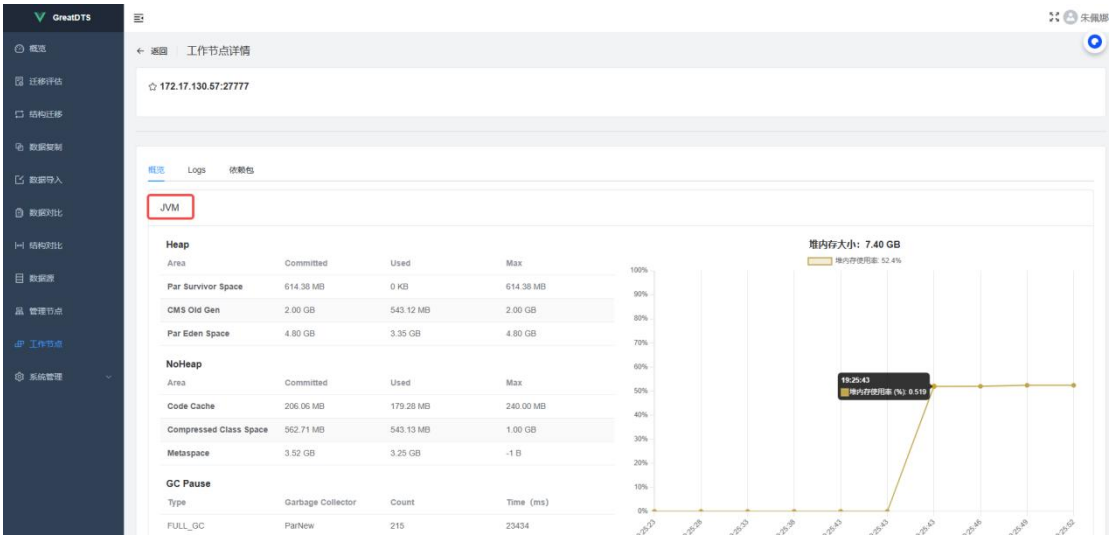
管理节点即 master 节点，工作节点即 worker 节点

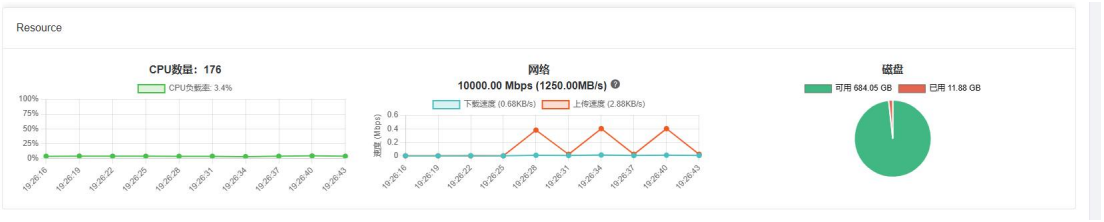
620-GA1 增加了对 master\worker 节点的信息采集，方便用户只管的查看 master\worker 节点的运行情况

点击 path 可以查看到详细的信息

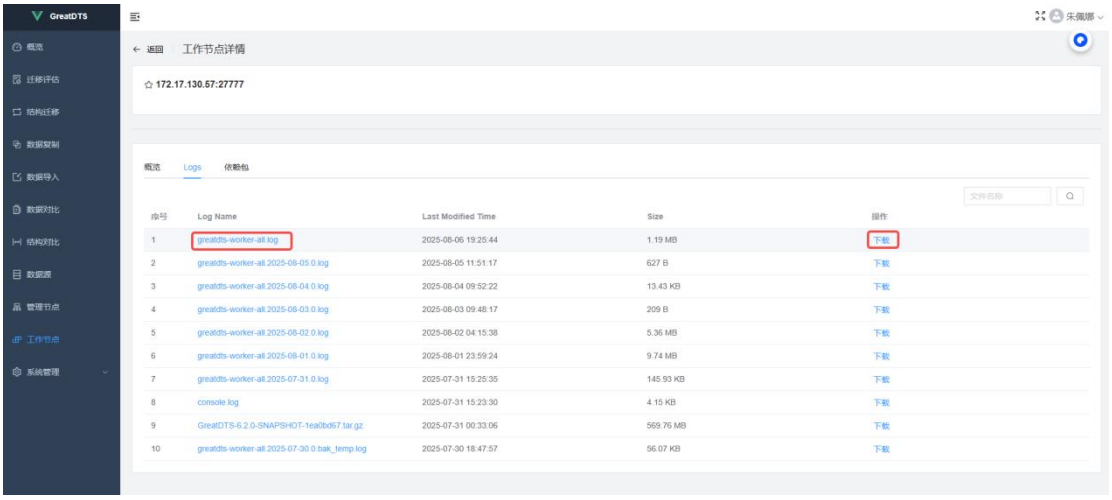


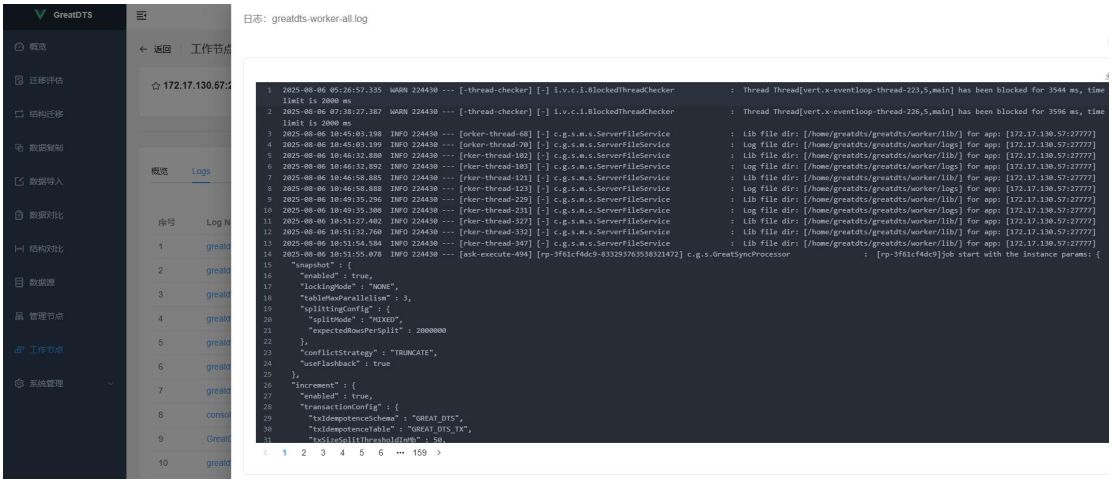
概览页面支持查看 JVM、资源的详细信息



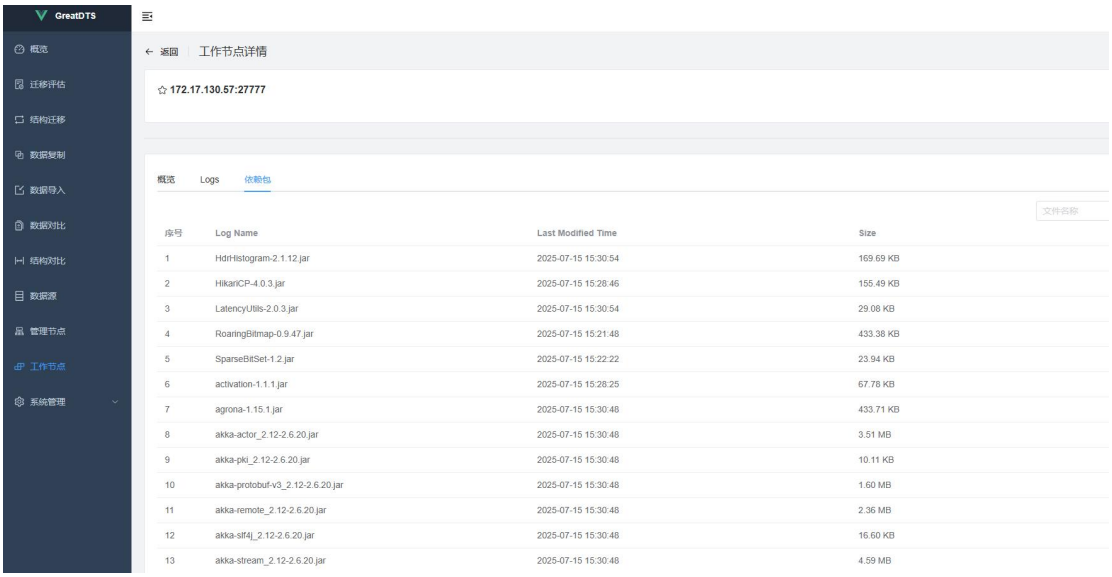


log 页面支持查看和下载 master\worker 节点的日志





支持查看 master\worker 节点的依赖包列表，依赖包列表，就是 lib 目录下所有内容



5. 日志收集

如果在 GreatDTS 使用过程中遇到问题，需要联系支持、开发人员排查。请参考如下步骤收集相关的日志信息。

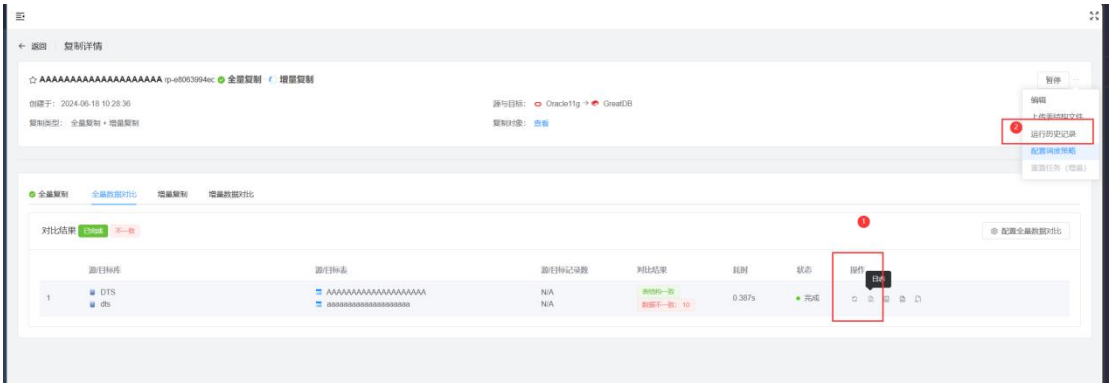
报错分任务报错和 DTS 服务报错

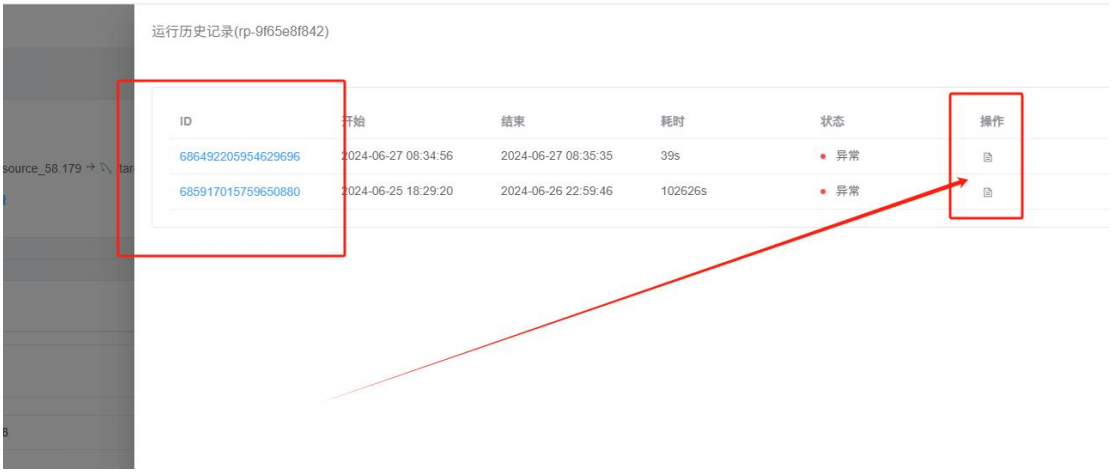
- 任务报错：任务运行中出现错误，DTS 服务并没异常。可直接在 UI 上查看\下载报错日志。
- DTS 服务报错：工作节点异常、页面上有明显的报错弹窗，需要进入 DTS 目录收集日志

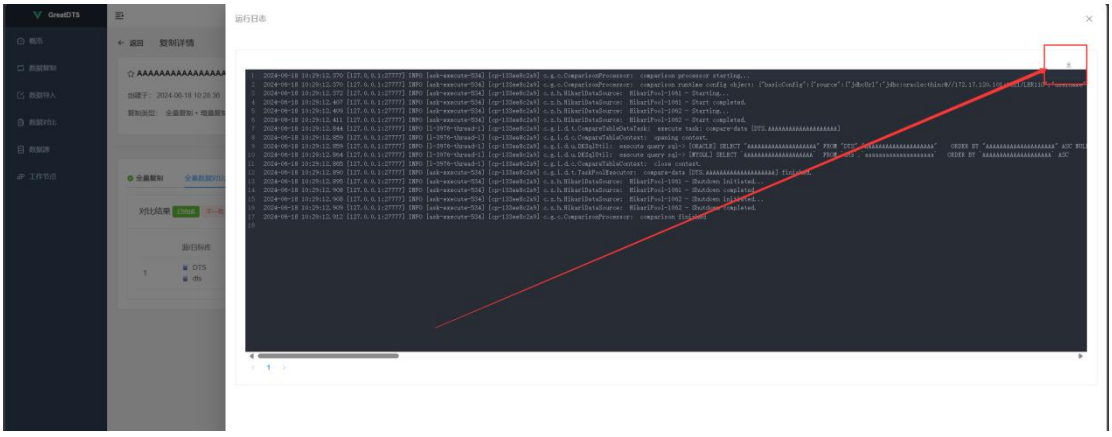
5.1 任务报错日志收集

1. 收集-**报错日志**

可在界面上直接查看和下载日志，提交该日志即可。注：下载按钮比较小。

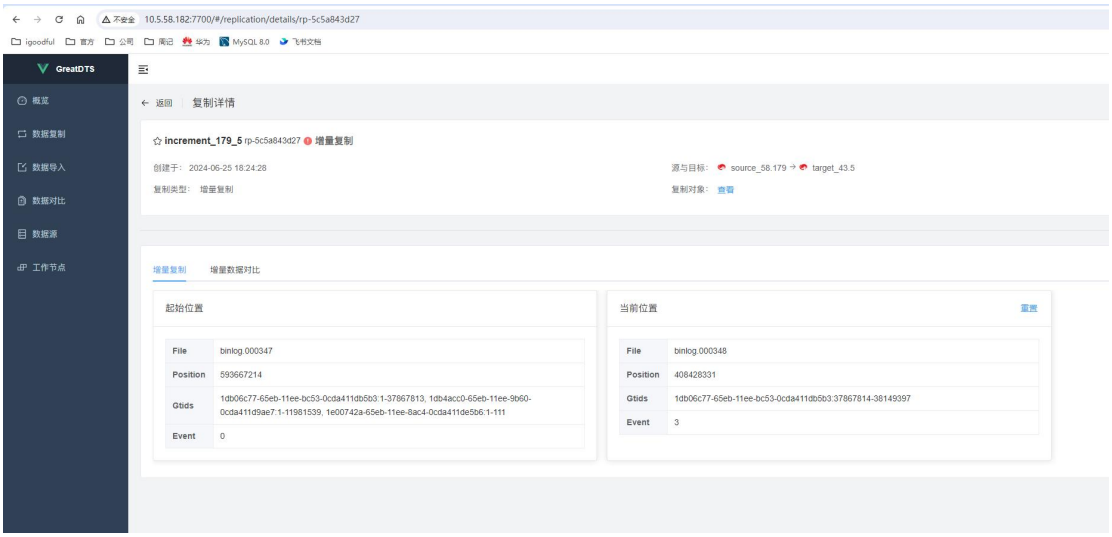






2. 收集-任务运行界面截图

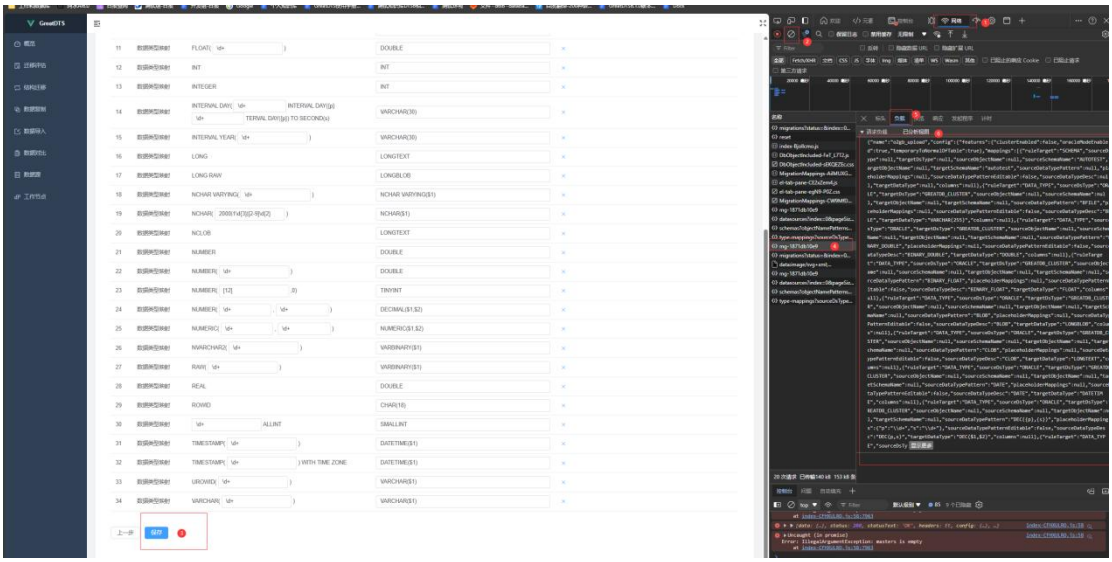
需要收集任务运行界面页面截图，以便观察任务运行情况和复制任务的数据位置



3. 收集-任务配置信息

需要收集任务的配置信息。点击任务的编辑按钮，到保存任务的界面。

打开 开发者设置-进入网络（Network）- 点击页面上的保存按钮->收集 DTS 发送的请求（名字是任务编号）



5.2 服务报错日志收集

1. 日志文件目录结构

```
YAML
#maser 节点日志
```

```
├──── logs
|     ├──── console.log  #控制台输出日志
|     ├──── greatdts-server-application.log #全量日志
|     ├──── greatdts-server-error.log #错误日志
|     └──── greatdts-server-web.log #前端日志，记录前端请求信息

#worker 节点日志
├──── logs
|     ├──── console.log #控制台输出日志
|     └──── greatdts-worker-all.log #全量日志
```

2. 内存溢出堆栈收集

如果在日志文件里看到内存溢出 OOM 相关的日志，需要修改 java 的启动脚本，打开相关参数（影响性能）。以便在下次 OOM 时生成堆栈文件方便开发排查：

在 startup.sh 的启动参数里添加 `-XX:+HeapDumpOnOutOfMemoryError`

会在 bin 目录下生成相关文件（通常为几个 G），提交给开发即可

6. FAQ

1.611-GA3 及以下版本当源（oracle）端和目标端（greatdb 或 mysql）配置了字段映射时会触发 bug，导致回流数据复制任务停止，报错为：“Increment synchronization failed with the error message: BatchUpdateException: ORA-00904: "xxxxx".invalid identifier”

解决方案：联系技术支持或对应的研发，获取当前 GreatDTS 的版本的修复 jar 包，替换一下 worker/lib 目录下的补丁包即可。

2.ORA-01555:Snapshot too old: rollback

全量同步时，配置的 SCN < 回滚段中最小的 scn。需要配置加大 ORACLE 的闪回空间，详情请咨询 Oracle DBA

3.Deadlock found when trying to get lock; try restarting transaction

在 GreatDB/MySQL 隔离级别为 repeatable-read 时，执行 insert on duplicated key update 由于间隙锁，可能导致死锁的问题。在全量复制时，可以考虑使用策略为【目标表存在存量数据的处理策略】- 【清空存量数据，重新写入】/ 【忽略存量数据，追加写入】

4.从 6.1.0 升到 6.1.1

6.1.1 的配置不兼容 6.1.0 的，所以最好的办法就是从头开始安装和配置 6.1.1 的版

本，只要配置元数据库的连接跟旧版本的一样旧的版本的数据就不会丢失。

4.6.1.1 之间的替换（既升级/回退）

简而言之，就是把旧的文件夹改名（也就是备份以防新的出问题还可以回退），把新的包正常解压，然后把旧的配置复制并覆盖到新的包下。

假设要被替换的 greatdts 目录在/opt/greatdts，新的包(假设名称为 GreatDTS-6.1.1-SNAPSHOT-aef1db33.tar.gz)也在/opt 目录下

1）停止和备份旧的版本：

```
Shell
/opt/greatdts/master/bin/stop.sh && /opt/greatdts/worker/bin/stop.sh &&
mv /opt/greatdts /opt/greatdts_backup
```

2）解压新包到/opt 下，解压完会有 greatdts 目录生成，进入该目录继续解压出 master 和 worker：

```
Shell
tar xvf /opt/GreatDTS-6.1.1-SNAPSHOT-aef1db33.tar.gz -C /opt && cd
/opt/greatdts/ && tar xvf ./greatdts-master-*.tar.gz && tar
xvf ./greatdts-worker-*.tar.gz
```

3）拷贝旧的配置文件覆盖新的

```
Shell
cp /opt/greatdts_backup/master/config/application.properties
/opt/greatdts/master/config/ && cp
/opt/greatdts_backup/worker/config/application.properties
/opt/greatdts/worker/config/
```

4）至此版本就替换好了。

5.Worker 显示下线/timeout/不可用

1）可能是 worker 由于 OOM（内存溢出）挂了

2）可能是 worker 的负载太大导致 Full GC 太久从而引发心跳没有及时发送到 master 从而被 master 判定为 timeout

3）可能是 worker 和 master 的时钟偏差太大（超过 1 分钟），导致 master 始终认为 worker 是 timeout 的状态

6.补丁包更新

将/master/lib 、/worker/lib 目录下的 jar 替换为相应的补丁包，重启 master、worker

SQLException: Lost connection to MySQL server during query

数据全量复制源端为 GreatDB Cluster 时

伴随着以下报错

```
Shell
Exception while closing JDBC connectionjava.sql.SQLException: Streaming result set com.mysql.cj.protocol.a.result.ResultsetRowsStreaming@1a6f2a26 is still active. No statements may be issued when any streaming result sets are open and in use on a given connection. Ensure that you have called .close() on any active streaming result sets before attempting more queries.
```

解决：尝试调大分布式集群 net_keep_alive_user_timeout 参数的值

参考：50000 => 86400000

GreatDB
万里数据库

联系我们 Contact Us

地址：北京市朝阳区CBD国际大厦7层701B
电话：400-032-7868
邮箱：sales@greatdb.com
网站：https://www.greatdb.com

北京万里开源软件有限公司

稳定 · 性能 · 易用