# 多功能任务编排器

【V1.0】

**详细设计说明书**

**2024年07月30日**

[多功能任务编排器 1](#_Toc174285497)

[1. 引言 4](#_Toc174285498)

[1.1. 编写目的 4](#_Toc174285499)

[1.2. 背景 4](#_Toc174285500)

[1.3. 定义 5](#_Toc174285501)

[1.4. 概念定义 5](#_Toc174285502)

[1.4.1. 流程模板 5](#_Toc174285503)

[1.4.2. 子流程模板 5](#_Toc174285504)

[1.4.3. 流程实体 5](#_Toc174285505)

[1.4.4. 子流程实体 5](#_Toc174285506)

[1.4.5. 全局流程数据流 6](#_Toc174285507)

[1.4.6. 子流程数据流 6](#_Toc174285508)

[1.4.7. 错误数据流 6](#_Toc174285509)

[1.5. 参考资料 6](#_Toc174285510)

[2. 程序系统结构 7](#_Toc174285511)

[3. 详细设计 7](#_Toc174285512)

[3.1. 程序简单描述 7](#_Toc174285513)

[3.2. 执行时序图 8](#_Toc174285514)

[3.2.1. 初始化 8](#_Toc174285515)

[3.2.2. 用户操作时系统中活动 9](#_Toc174285516)

[3.2.3. 完成后交付通知 9](#_Toc174285517)

[3.3. 主要类的说明和xml文件的配置 10](#_Toc174285518)

[3.3.1. Xml文件配置 10](#_Toc174285519)

[3.3.2. 核心类说明 12](#_Toc174285520)

[3.4. 测试报告 13](#_Toc174285521)

[3.4.1. 子任务注册 13](#_Toc174285522)

[3.4.2. 程序启动 14](#_Toc174285523)

[3.4.3. 编排任务执行结果 14](#_Toc174285524)

[3.5. 尚未解决的问题 15](#_Toc174285525)

[3.5.1. 暂时没有web图形界面 15](#_Toc174285526)

[3.5.2. 高并发下发任务时存在线程饥饿现象 16](#_Toc174285527)

[4. 使用手册 16](#_Toc174285528)

[4.1. Pom文件配置 16](#_Toc174285529)

[4.2. 创建子任务 16](#_Toc174285530)

[4.2.1. 步骤一： 16](#_Toc174285531)

[4.3. 启动应用自动初始化数据 17](#_Toc174285532)

[4.3.1. 启动代码如下 17](#_Toc174285533)

[4.4. 编排子任务 18](#_Toc174285534)

[4.4.1. 编排子任务脚本 18](#_Toc174285535)

[4.5. 调用接口直接任务 18](#_Toc174285536)

1. 引言
   1. 编写目的

此说明书在概要设计的基础上，对任务编排器内部具体逻辑、使用方式、测试方式、实现效果等方面分别进行了详细的描述和记录。在以下的各模块文档中，将对V1.0版本的任务编排器系统做详细的说明。

主要工作包括：根据软件需求说明所描述的功能、数据、运行、环境，依照概要设计说明书中描述的总体设计、结构设计、进行逐个模块的程序描述（包括各模块的功能、输入输出、算法、程序逻辑、接口等）。

软件开发的小组实现成员、使用人员应该参考此说明书、从而进行软件的代码编写和测试以及使用。

* 1. 背景

随着计算机资源的逐渐发展，为了将计算、网络、存储等抽象资源最大利用化，计算机将资源进行整合变为云，将互联网推进了云计算时代，而云计算中存在分钟级的虚拟机云，和秒级的容器云等资源都需要存在一个“管理者“进行资源的调度和协调，因此诞生了云管理平台。云管理平台和互联网不同之处在于，云资源管理平台面临的挑战并非高并发和高性能优先，而是具有稳定和可用以及步骤可视化和步骤可操作化的能力。

云管理平台在虚拟机和容器等资源分配上起到了非常关键的作用，该平台处理的业务特点为业务流程长，并发量不高，为了完成用户链路需要执行多步骤，并且每一步骤需要大量时间（分钟级），因此必须采用异步的思想处理该业务，而异步的业务同样需要在不确定的网络稳定性和业务资源缺失的风险下保证平台的可用性（可用性代表无论如何都能够给与用户一个合理的结果，包括合理的报错结果），而在资源分配的过程中通常会遇到不可抗拒的因素而导致中途环节异常而导致整个流程终止，无法继续和重试，通常只能重新开始并且无法回收已经分配的资源，该痛点不仅浪费计算资源更对运维人员造成巨大的压力。因此针对该场景，团队推出了轻量级的任务编排引擎，不仅能够监控整个流程的启停并且还能够对整体流程进行监控。

* 1. 定义

多功能任务编排器是一款使用java语言编写的轻量级业务流程编排引擎，该引擎提供了长任务的增删改查接口以及版本控制能力，并能够灵活定义任务中的子任务来执行整体业务流程。引擎不负责业务流程中的校验以及逻辑判断工作，其主要管理和监控每一个子任务的执行过程，保存子任务执行过程的输入输出快照，并基于此能力衍生出各子任务重试，回滚，跳过，暂停和停止的重要功能，即使在业务代码出错的情况下，引擎仍然能够记录子任务执行的全部过程以及快照，并给用户提供可以选择的控制能力。

* 1. 概念定义
     1. 流程模板

表名：FLOW\_TPL

记录一个完整的工作流程，该表记录了任务流程的各种详细信息，包括版本、名称、描述、类型、创建时间以及历史完成时间等

* + 1. 子流程模板

表名：FLOW\_sub\_tpl

记录工作流程模板中需要执行个各项子任务实体，包括子任务模板，子任务名称、类型等模板信息

* + 1. 流程实体

类似于进程，从流程模板中实例化一个流程实其中记录执行任务的状态，对应的模板、以及创建和已经运行的时间等信息。

* + 1. 子流程实体

类似于线程，从子流程模板中实例化子流程实体，其中记录当前子任务状态，运行时间等详细信息。

* + 1. 全局流程数据流

完整的流程用到的全局变量，比如固定当前任务中不会改变的任务变量，能够减少子任务之间的冗余数据流传递，类似于上下文context

* + 1. 子流程数据流

表示子流程之间传递的入参以及返回值的记录，不仅用于任务之间的数据交互，还用于子流程审计和重试回滚等。

* + 1. 错误数据流

记录子任务执行过程中出现的业务错误或引擎错误信息，便于问题排查和可视化展示错误原因

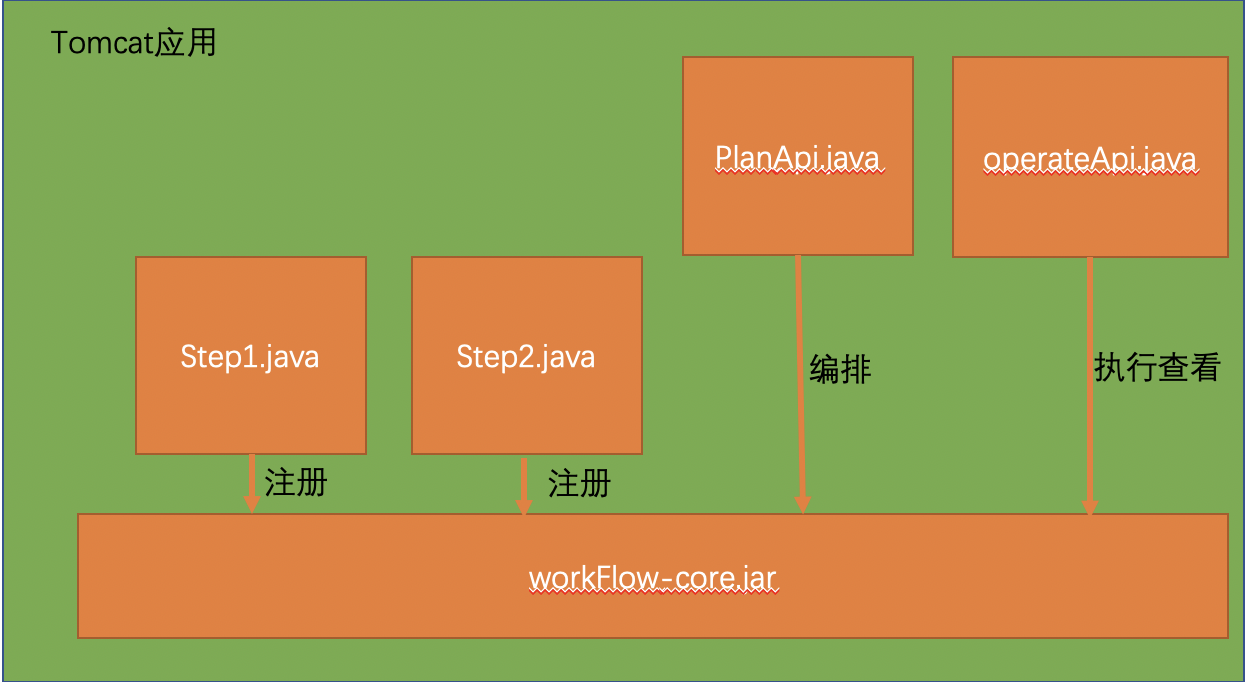
* 1. 参考资料

《Flowable框架实战》

《BPMN2.0 xml规范手册》

《mybatis-plus用户手册》

1. 程序系统结构



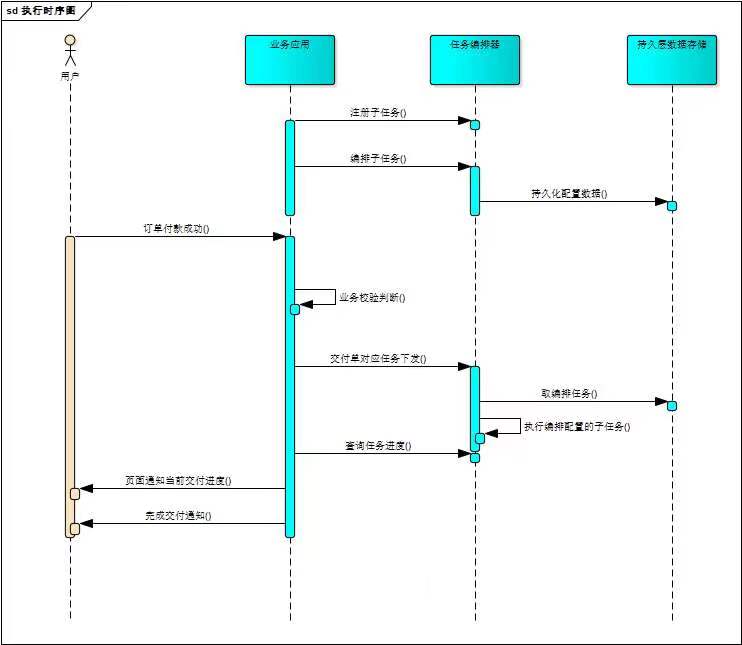
图一

如图一所示，该图展示了workFlow引擎在整个tomcat应用中所处的位置和能够发挥的功能，其中workFlow-core.jar中包含了多功能任务编排器的所有核心代码，该代码提供了业务子步骤代码的注册能力，编排能力，任务操作以及任务快照监控查看能力。Step1.java和Step2.java文件由用户自行撰写，其中包含用户期望的业务处理代码。PlanApi.java为用户编写的调用引擎编排功能接口代码。

1. 详细设计
   1. 程序简单描述

多功能任务编排器是一款针对长流程任务场景设计的一款任务执行器和任务监控器，其功能主要有：自动化注册任务，自动化执行任务，监控任务，支持手动操作回滚、重试、跳过步骤等。

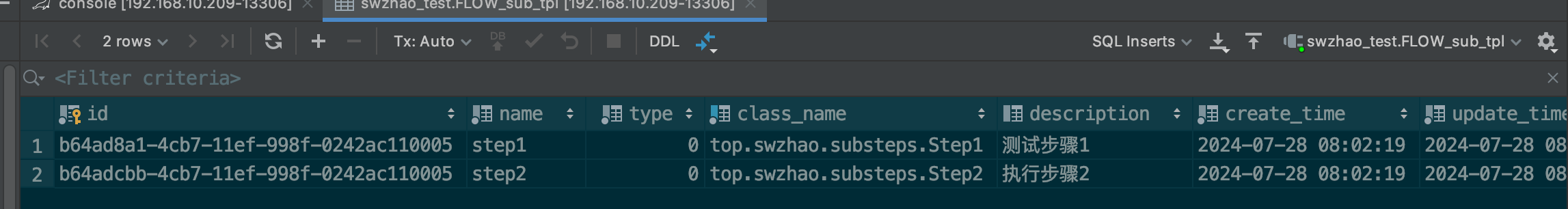
* 1. 执行时序图



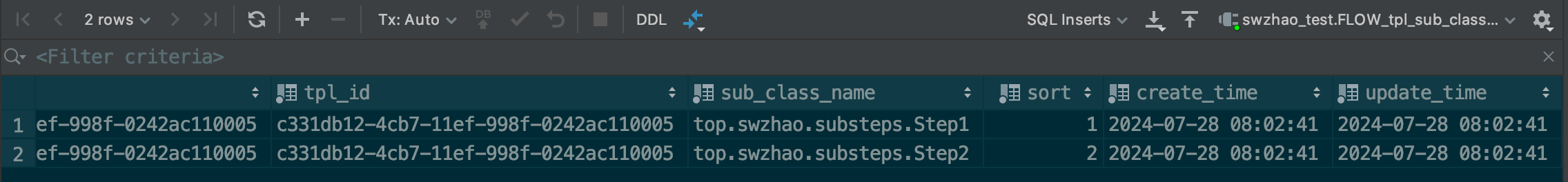
* + 1. 初始化

注：该过程为应用上线前，后台运维或开发人员进行手工配置完成

1. 注册子任务：开发人员将业务流程执行的逻辑按照业务特性分为若干步骤，并按照任务编排器协议进行编码，使用注解@FlowDiscription进行任务注册，完成后，应用打包发布到用户环境，此时数据库FLOW\_sub\_tpl中已经存储开发人员注册的步骤记录，如下图：



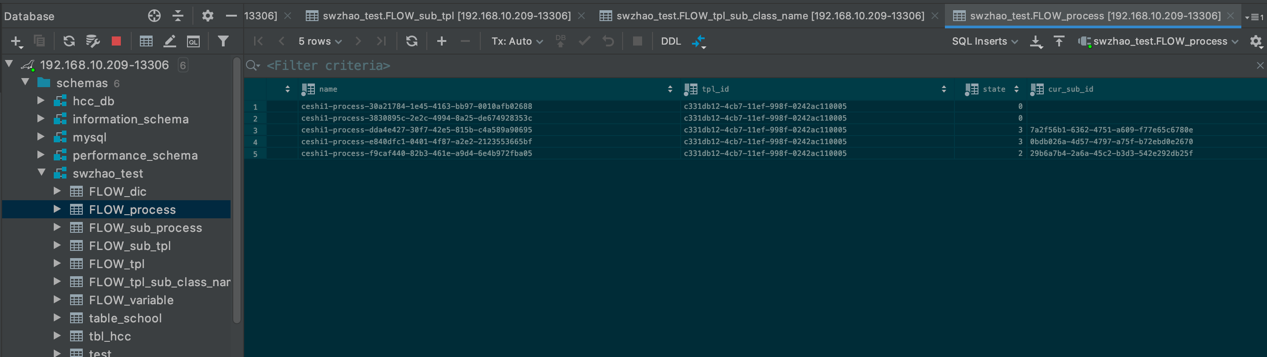
1. 编排子任务：编排子任务工作可以由运营，产品，运维按照业务进行自由编排，编排完成后，任务编排器将记录用户的编排配置，如下：



至此，业务系统即可对外提供交付服务

* + 1. 用户操作时系统中活动

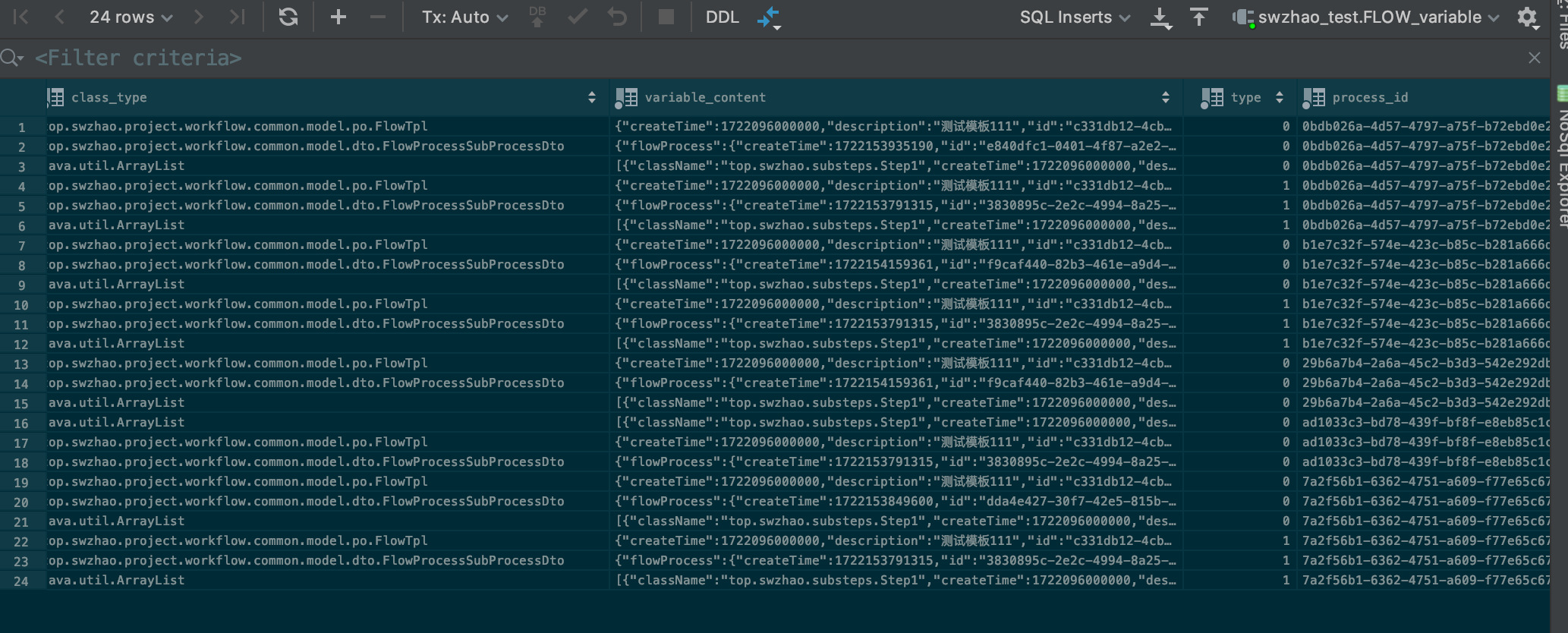
1. 用户付款成功，业务系统进行扣款，库存，资源，授权等一系列上层业务判断后，开始下发交付任务
2. 下发交付任务给编排器，通过订单内容找到映射的任务编排配置模板（由运维和产品编排配置），任务编排器将任务上下文初始化后交给执行线程执行，并立刻返回下发成功给业务应用，此时编排器持久层中写入当前交付任务记录，如下图：



1. 执行期间，业务系统可随时查看任务状态，并实时通知用户，播报交付进度（可以手机、短信、邮箱等方式，有业务应用自行实现）
   * 1. 完成后交付通知

任务完成后，业务应用可自行检查主任务status字段是否为结束状态并及时通知用户，交付是否完成或者交付是否失败

若任务失败，用户可选择继续执行还是跳过、回滚等操作，业务应用发下执行给应用编排，应用可以直接执行各步骤，各步骤执行过程输入输出存储如下：



* 1. 主要类的说明和xml文件的配置
     1. Xml文件配置

1. 序列化工具类，用户序列化存储执行过程用到的数据结构:

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>${fastjson.version}</version>

</dependency>

1. 数据库持久层驱动和ORM框架能力配置

<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-boot-starter</artifactId>

<version>${mybatis-plus.version}</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>${mysql.connector.version}</version>

</dependency>

1. Spring基本注入bean的能力依赖：

<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-core</artifactId>  
 <version>${spring.version}</version>  
 <scope>compile</scope>  
</dependency>  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-beans</artifactId>  
 <version>${spring.version}</version>  
 <scope>compile</scope>  
</dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>${spring.version}</version>  
 <scope>compile</scope>  
</dependency>  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-web</artifactId>  
 <version>${spring.version}</version>  
 <scope>compile</scope>  
</dependency>

1. 测试工程能力

<!--test-->  
<dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>${junit.version}</version>  
</dependency>  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-test</artifactId>  
 <version>${spring.test.version}</version>  
</dependency>

* + 1. 核心类说明

注：核心类中的方法仅选择典型代表性方法进行解释，剩余方法源码中存在注释

|  |  |  |  |
| --- | --- | --- | --- |
| 类名 | 方法 | 功能描述 | 备注 |
| top.swzhao.project.workflow.  common.engine.OmpFlowEngine | 无 | 唯一暴露给客户端的引擎入口（最重要的核心类） |  |
| start | 基于某个模板id启动一个流程 |  |
| getFlowTplService | 获取模板增删改查能力 |  |
| rollback | 回滚某个任务 |  |
| top.swzhao.project.workflow  .core.init.InitEngine | Run | 1、检测应用中的表结构是否完全， 如果不完整需要补全所有表项 2、加载项目中实现了OmpFlow的实现类 3、将所有实现类名作为索引对数据库表进行检索 4、如果没有对应的子模板的类名则数据库 insert into on duplicate key update 5、输出引擎logo |  |
| top.swzhao.project.workflow.  common.annotation.FlowDescription | 无 | 注册子任务类，注释子任务类并在注释中为子任务取名和标记，最终程序启动时将自动扫描注解并入库数据 |  |

* 1. 测试报告
     1. 子任务注册

业务测试代码：

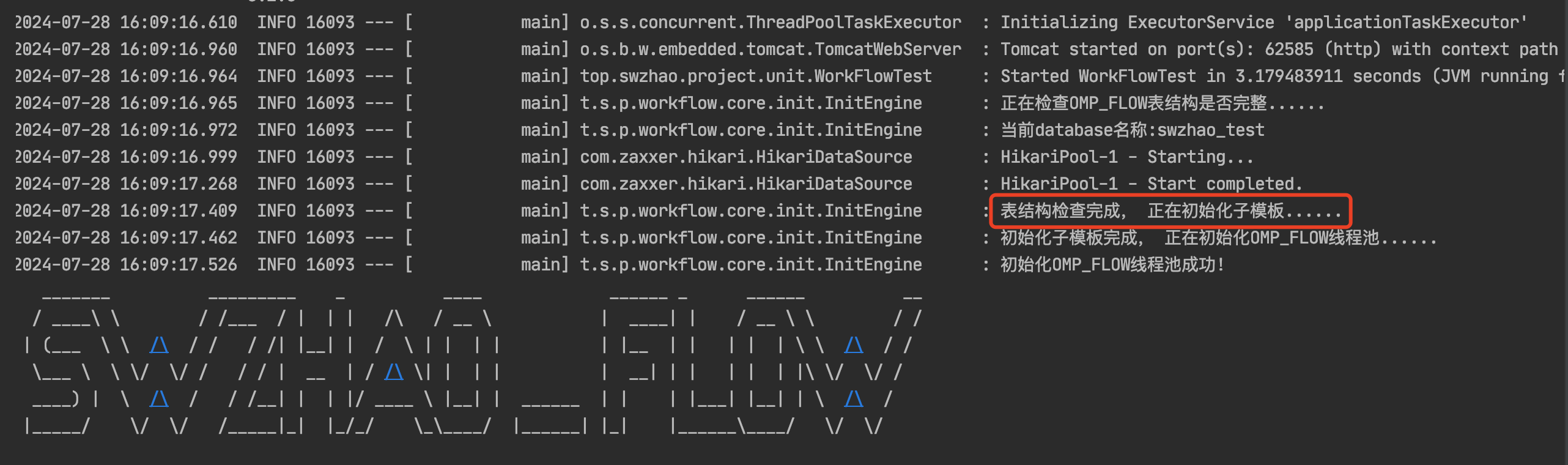
步骤一：



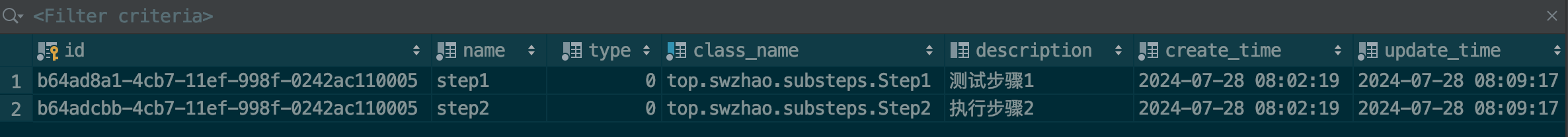
步骤二：



1. 子任务注册日志

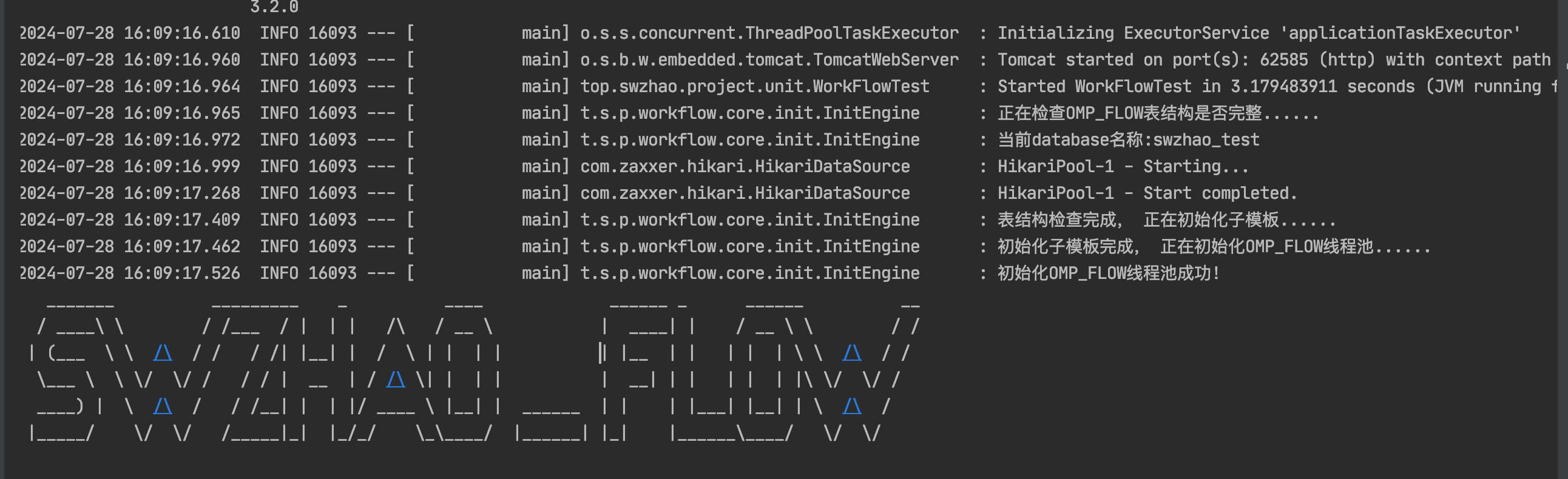


1. 子任务模板数据库写入截图



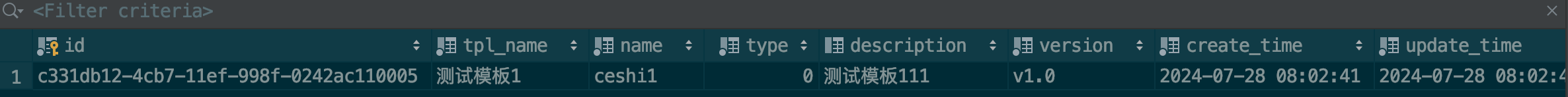
* + 1. 程序启动

1. 程序启动日志



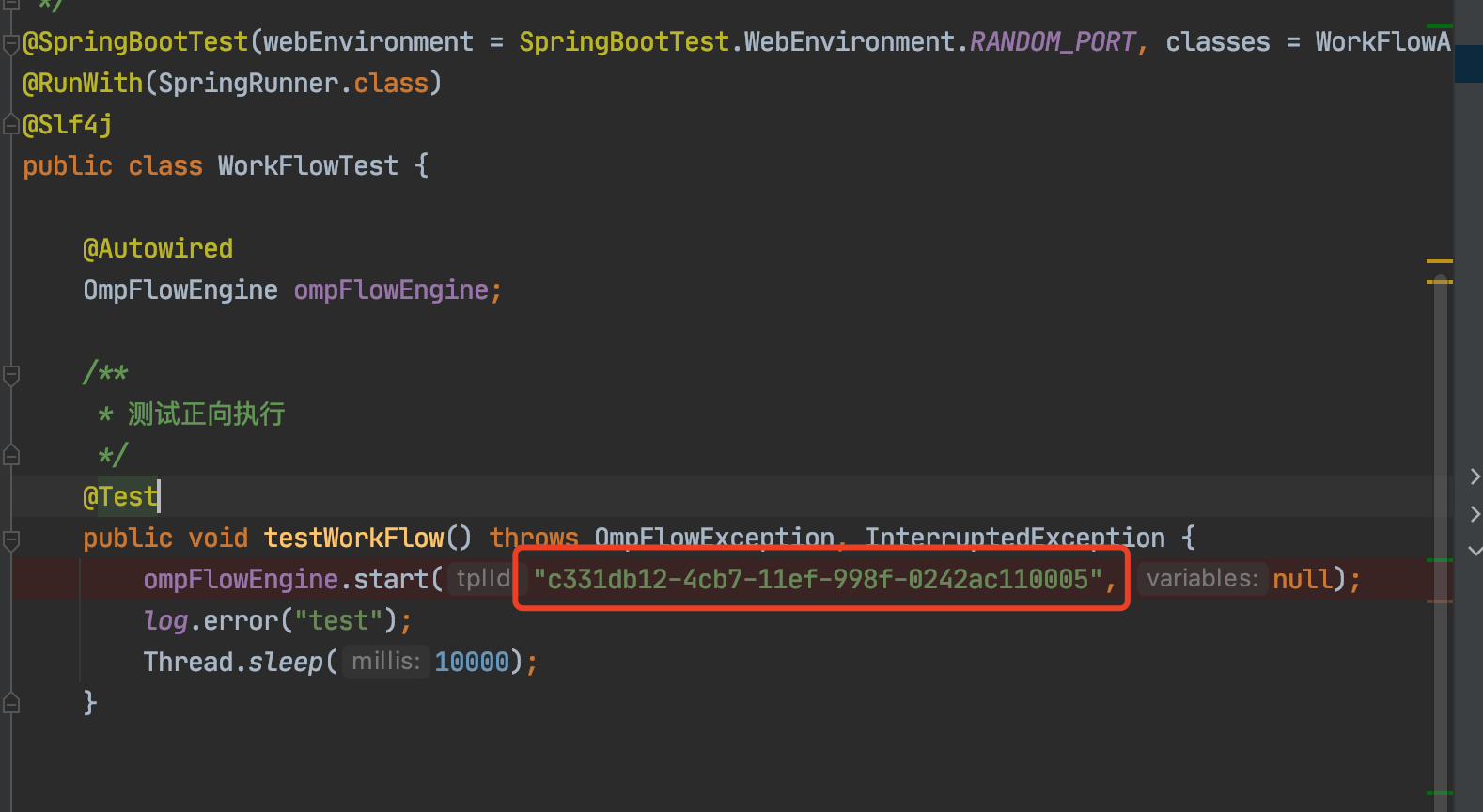
1. 数据库初始化数据截图

主任务模板数据写入如下：



* + 1. 编排任务执行结果

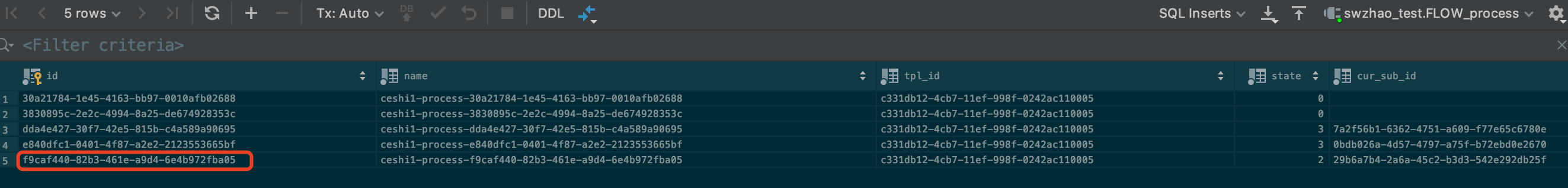
执行模板id为c331db12-4cb7-11ef-998f-0242ac110005



1. 执行结果日志

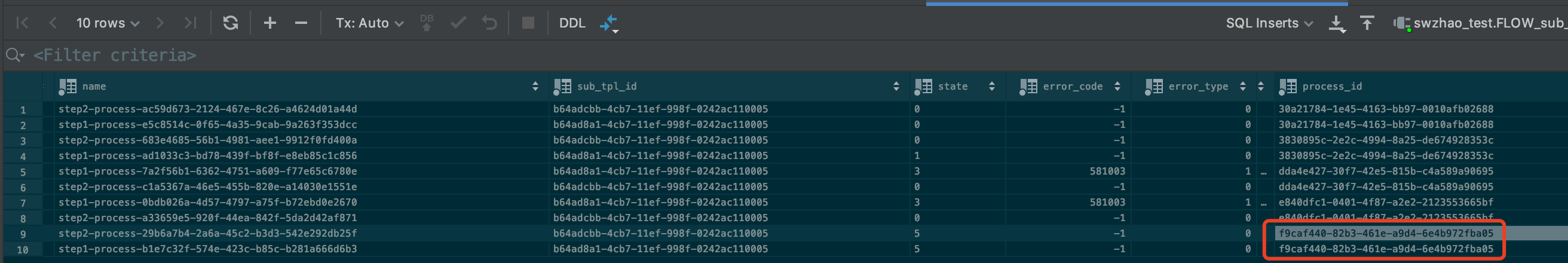


1. 父任务数据写入截图

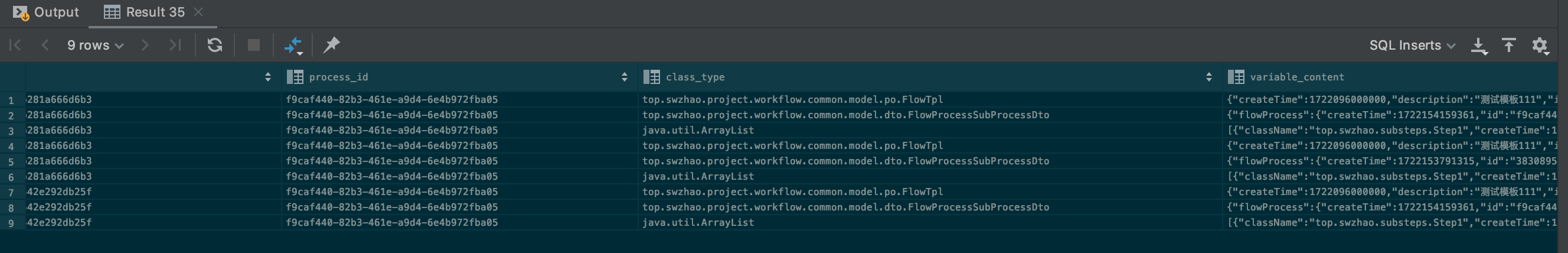


1. 子任务执行数据截图

父任务（id：f9caf440-82b3-461e-a9d4-6e4b972fba05） 下的子任务数据



1. 执行变量快照截图



* 1. 尚未解决的问题
     1. 暂时没有web图形界面

目前任务编排器并未提供后台web图形界面，仅提供数据增删改查接口，统一由类：OmpFlowEngine提供，应用程序可自行设计前端界面，直接使用接口即可提供界面支持。

* + 1. 高并发下发任务时存在线程饥饿现象

目前编排器并无多种执行优先级策略，仅支持FCFS策略，后续增加LRU、响应比策略来增加执行效率

1. 使用手册
   1. Pom文件配置

<dependency>  
 <groupId>top.swzhao.project</groupId>  
 <artifactId>workflow-core</artifactId>  
</dependency>

注：

仅依赖workflow-core即可，剩余部分将会间接依赖进来

* 1. 创建子任务

以下为步骤示例，实际场景可自行撰写业务代码

* + 1. 步骤一：

@FlowDescription(description = "测试步骤1")  
@Slf4j  
@Service  
public class Step1 implements OmpFlowable {  
  
  
 @Override  
 public void start(TempFlowParam tempFlowParam, GlobalFlowParam globalFlowParam) {  
 log.warn("flowable Step1 start！");  
 }  
  
 @Override  
 public void rollback(TempFlowParam tempFlowParam, GlobalFlowParam globalFlowParam) {  
 log.warn("flowable Step1 rollback！");  
 }  
}

步骤二：

@FlowDescription(description = "执行步骤2")  
@Slf4j  
@Service  
public class Step2 implements OmpFlowable {  
 @Override  
 public void start(TempFlowParam tempFlowParam, GlobalFlowParam globalFlowParam) {  
 log.warn("flowable Step2 start！");  
 }  
  
 @Override  
 public void rollback(TempFlowParam tempFlowParam, GlobalFlowParam globalFlowParam) {  
 log.warn("flowable Step2 rollback！");  
 }  
}

* 1. 启动应用自动初始化数据
     1. 启动代码如下

@SpringBootApplication

@Slf4j

public class WorkFlowApplicatin {

public static void main(String[] args) {

try {

SpringApplication.run(WorkFlowApplicatin.class, args);

}catch (Exception e) {

log.error("", e);

}

}

}

* 1. 编排子任务
     1. 编排子任务脚本

-- 对子任务进行编排的sql数据  
-- 刷入测试模板以及模板子模板关系  
set @tplId = UUID();  
INSERT INTO swzhao\_test.FLOW\_tpl (id, tpl\_name, name, type, description, version, create\_time, update\_time)  
VALUES (@tplId, '测试模板1', 'ceshi1', 0, '测试模板111', 'v1.0', now(), now());  
INSERT INTO swzhao\_test.FLOW\_tpl\_sub\_class\_name (id, tpl\_id, sub\_class\_name, sort, create\_time, update\_time)  
VALUES (UUID(), @tplId, 'top.swzhao.substeps.Step1', 1, now(), now());  
INSERT INTO swzhao\_test.FLOW\_tpl\_sub\_class\_name (id, tpl\_id, sub\_class\_name, sort, create\_time, update\_time)  
VALUES (UUID(), @tplId, 'top.swzhao.substeps.Step2', 2, now(), now());

* 1. 调用接口直接任务

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM\_PORT, classes = WorkFlowApplicatin.class)  
@RunWith(SpringRunner.class)  
@Slf4j  
public class WorkFlowTest {  
  
 @Autowired  
 OmpFlowEngine ompFlowEngine;  
  
 /\*\*  
 \* 测试正向执行  
 \*/  
 @Test  
 public void testWorkFlow() throws OmpFlowException, InterruptedException {  
 ompFlowEngine.start("c331db12-4cb7-11ef-998f-0242ac110005", null);  
 log.error("test");  
 Thread.sleep(10000);  
 }  
  
  
 /\*\*  
 \* 测试回滚  
 \*/  
 @Test  
 public void testWorkFlowRollBack() {  
  
 }  
  
  
}