Activiti(Flowable)流程克隆（运行时实例克隆）解决方案

一种订单制造中分裂与合并的核心技术实践

作者：长沙大东家 xiaohelong2005@126.com 2017-11-24 20:24 长沙

20180425 完善性更新

1. 背景

订单可视化系统是单位的一套基于业务流程引擎技术的智能制造信息系统，在公司领导的战略部署下，完全自主开发的一套适用于多品种、小批量订单式制造型企业。系统以对订单的全生命周期进行管控。通过对订单的售前、生产、售后及过程中产生的异常等进行记录并分析。再以此为基础，对各个业务环节逐步细化，由于是对一个企业进行全方位流程再造，涉及到企业所有方面，持续时间较长，并且随着企业的发展而发展的一套系统。

在对流程引擎技术选择中，我们选择了 Activiti(现在作者重新 Fok 出叫 Flowable 的项目),开源、稳定，社区活跃。但外国人开发的流程系统并没有考虑到中国的实际使用情况，如驳回、任意跳转等。更不用说会想到中国制造型企业的特色流程需求，如订单随时分裂与合并。

2. 问题提出

订单在流程的执行过程中，企业中会在所有可能的环节出现订单分裂的需求。如因为采购的物料只来的一部分，客户同意先提走一部

分；客户在设计环节或制造前突然取消几套设备都有可能；即使在机器制造完毕后，还有可能分不同批次发送，补生产之类。Activiti 并不支持理想中类似于管道中跑包裹的情况，对，最理想的方式就是隧道中跑小汽车的模式，每一台设备每一个一个编号，再对编号进行分组成一个订单便可以解决这个问题。但 Activiti 很明显不支持这种模式，自己开发一个框架时间和成熟度上也不合适。只能在 activiti 基础上进行。

临时的解决方案便是将原来的流程挂起，再新开几个对应的流程，再跳转至原来的任务节点。这么做临时性解决了一些问题，但也带来了两个实际的问题：一、新的流程并没有携带以前的历史信息；二、跳转只能单线，当有并行多线路时，则只能跳到原来的一根线上，其它的线会断层。所以克隆是从最初至现在一直想要的实施方案，克隆可以完整地解决这些问题，订单拆分时可以克隆多份并配套至业务订单即可。因为时间与技术的熟悉程度，所以用了一点时间研究了 Activiti 机制，决定从直接更改数据库中的数据为总体思路。即将该流程所有信息复制一问，换用新的 ID。

为什么要自己做呢？因数无论是哪里都找不到资料，只能靠自己来解决此事。

3. 第一次试错

在起初实施的时侯，我试图通过表之间的关系、字段之间的关系，逐层逐表进行替换。但在逐步的深入中，子流程、并行、递归及相互

字段的更新等导致逻辑越来越复杂，最终不得不放弃此种实施思路。

4. 最终思路

在某一个瞬间，来了一个灵感，想到以前在做网页时全局将 UUID 替换成指定单词的经验，我们在 Java 代码中应用这种全局文本替换结合 UUID 的唯一性与特殊性，最终思路便是将该流程实例的所有信息全部读入，将每一个 ID 列入一个集合中，再配套分配一个新的 ID 反向存入数据，得以解决，逻辑思路上也很解单。

研究与实践就是研究需要试错 10 次，100 次才能找到正确的一次方式。

具体思路如下：

1. 首先读取所有流程相关的记录，只要处理 act_hi_*,act_ru_*，act__ge_bytearray。
2. 将所有 id 字段读入(有一些 id 字段要排除，如 procDefId, bytearrayId)集合，如果是初次进入集合给定一个对应的新 id,如果是已存在，则直接从集合中读取该 id 放入此记录中。
3. 读取完毕后，直接用新 id 替换对应的数据。
4. 批量存入数据库，流程克隆完成。(在 insert 时，act_ru_*的需要临时将数据库外键关闭)
5. 检测是否执行完毕，结束

对于子流程，进行递归调用处理。对于程序中没有克隆的表，大家参照这个思路进行添加处理，因为目前这些对我们够用了。

6. 其它

将 act_ru_variable 中对 act_ge_bytearray 中的外键进行修改，防止更新出错。删除时 set null,更新时 set null。

5. 最佳实践

5.1. Activiti(Flowable)5.21 的数据表(Mysql and mybatis)基础（重点关注 act_hi_*,act_ru_*的中 id 相关字段）

5.1.1. 基础说明

参考地址(reference)：http://lucaslz.com/2016/11/15/java/activiti/activiti-db-5-22/

ACT_RU_*：‘RU’表示 runtime，运行时表-RuntimeService。这是运行时的表存储着流程变量，用户任务，变量，职责（job）等运行时的数据。Activiti 只存储实例执行期间的运行时数据，当流程实例结束时，将删除这些记录。这就保证了这些运行时的表小且快。

ACT_HI_*:

’HI’表示 history，历史数据表，HistoryService。就是这些表包含着流程执行的历史相关数据，如结束的流程实例，变量，任务，等等

5.1.2. 表摘要

ACT_HI_ACTINST	历史节点表
ACT_HI_ATTACHMENT	历史附件表
ACT_HI_COMMENT	历史意见表
ACT_HI_DETAIL	历史详情表，提供历史变量的查询
ACT_HI_IDENTITYLINK	历史流程人员表
ACT_HI_PROCINST	历史流程实例表
ACT_HI_TASKINST	历史流程任务表

ACT_HI_VARINST	历史变量表
ACT_RU_EVENT_SUBSCR	throwEvent、catchEvent 时间监听信息表
ACT_RU_EXECUTION	运行时流程执行实例表
ACT_RU_IDENTITYLINK	运行时流程人员表，主要存储任务节点与参与者相关信息
ACT_RU_JOB	运行时定时任务数据表
ACT_RU_TASK	运行时任务节点表
ACT_RU_VARIABLE	运行时流程变量数据表

5.1.3. 表结构说明(关键 ID 字段加粗,要包括本身的 ID_号)

5.1.3.1. ACT_HI_ACTINST（历史节点表）

历史活动信息。这里记录流程流转过的所有节点，与 HI_TASKINST 不同的是，taskinst 只记录 usertask 内容。

表结构说明

字段名称	描述	数据类型	Nullable	取值说明
ID_	**主键**	**varchar(64)**	**NO**	**主键 ID**
PROC_DEF_ID_	流程定义 ID	varchar(64)	NO	流程定义 ID
PROC_INST_ID_	**流程实例 ID**	**varchar(64)**	**NO**	**流程实例 ID**
EXECUTION_ID_	**执行实例 ID**	**varchar(64)**	**NO**	**执行实例 ID**
ACT_ID_	节点 ID	varchar(255)	NO	节点定义 ID
TASK_ID_	**节点实例 ID**	**varchar(64)**	**YES**	**默认值 NULL，其他节点类型实例 ID 在这里为空**
CALL_PROC_INST_ID_	**调用外部的流程实例 ID**	**varchar(64)**	**YES**	**默认值 NULL，调用外部流程的流程实例 ID**
ACT_NAME_	节点名称	varchar(255)	YES	默认值 NULL，节点定义名称
ACT_TYPE_	节点类型	varchar(255)	NO	如 startEvent、userTask
ASSIGNEE_	签收人	varchar(255)	YES	默认值 NULL，节点签收人
START_TIME_	开始时间	datetime(3)	NO	version 版本，2016-11-15 11:30:00
END_TIME_	结束时间	datetime(3)	NO	默认值 NULL，2016-11-15 11:30:00
DURATION_	耗时	bigint(20)	YES	默认值 NULL，毫秒值
TENANT_ID_	租户标识	varchar(255)	YES	默认值``，

5.1.3.2. ACT_HI_ATTACHMENT（历史附件表）

历史附件表。

表结构说明

字段名称	描述	数据类型	Nullable	取值说明
ID_	**主键**	**varchar(64)**	**NO**	**主键 ID**
REV_	乐观锁	int(11)	YES	默认值 NULL，version 版本

USER_ID_ 用户 ID varchar(255) YES 默认值 NULL，用户标识
NAME_ 名称 varchar(255) YES 默认值 NULL
DESCRIPTION_ 描述 varchar(4000) YES 默认值 NULL
TYPE_ 类型 varchar(255) YES 默认值 NULL，附件类型
TASK_ID_ 节点实例 ID varchar(64) YES 默认值 NULL
PROC_INST_ID_ 流程实例 ID varchar(64) YES 默认值 NULL，
URL_ URL 附件地址varchar(4000) YES 默认值 NULL，附件地址
CONTENT_ID_ 字节表的 ID varchar(64) YES 默认值 NULL, ACT_GE_BYTEARRAY 的 ID
TIME_ 乐观锁 datetime(3) YES 默认值 NULL

5.1.3.3. ACT_HI_COMMENT（历史意见表）

历史意见表

表结构说明

字段名称 描述 数据类型 Nullable 取值说明
ID_ 主键 varchar(64) NO 主键 ID
TYPE_ 类型 varchar(255) YES 类型：event（事件）、comment（意见）
TIME_ 时间 datetime(3) NO 填写时间
USER_ID_ 用户 ID varchar(255) YES 填写人 ID
TASK_ID_ 节点实例 ID varchar(64) YES 默认值 NULL
PROC_INST_ID_ 流程实例 ID varchar(64) YES 默认值 NULL
ACTION_行为类型 varchar(255) YES 默认值 NULL，值为下列内容中的一种：AddUserLink、DeleteUserLink、AddGroupLink、DeleteGroupLink、AddComment、AddAttachment、DeleteAttachment
MESSAGE_ 基本内容 varchar(4000) YES 默认值 NULL，用于存放流程产生的信息，比如审批意见
FULL_MSG_ 全部内容 longblob YES 附件地址

5.1.3.4. ACT_HI_DETAIL（历史详情表）

历史详情表：流程中产生的变量详细，包括控制流程流转的变量，业务表单中填写的流程需要用到的变量等。

表结构说明

字段名称 描述 数据类型 Nullable 取值说明
ID_ 主键 varchar(64) NO 主键 ID
REV_ 乐观锁 int(11) YES 默认值 NULL，version 版本
TYPE_ 类型 varchar(255) NO 默认值 NULL，表单：FormProperty，参数：VariableUpdate
PROC_INST_ID_ 流程实例 ID varchar(64) YES 默认值 NULL，附件类型

EXECUTION_ID_ 执行实例 ID varchar(64) YES 默认值 NULL，附件类型
TASK_ID_ 节点实例 ID varchar(64) YES 默认值 NULL，附件类型
ACT_INST_ID_ 节点实例 ID varchar(64) YES 默认值 NULL，附件类型
NAME_ 名称 varchar(255) NO 默认值 NULL，附件类型
VAR_TYPE_ 参数类型 varchar(255) YES 默认值 NULL， jpa-entity、boolean、bytes、serializable(可序列化)、自定义 type(根据你自身配置)、CustomVariableType、date、double、integer、long、null、short、string,
TIME_ 时间 datetime(3) NO 默认值 NULL，创建时间
BYTEARRAY_ID_ 字节表 ID varchar(64) YES 默认值 NULL, ACT_GE_BYTEARRAY 表的 ID
DOUBLE_ Double double YES 默认值 NULL，存储变量类型为 Double
LONG_ Long bigint（20) YES 默认值 NULL，存储变量类型为 long
TEXT_ Text varchar(40000) YES 默认值 NULL，存储变量值类型为 String
TEXT2_ Text varchar(40000) YES 默认值 NULL，此处存储的是 JPA 持久化对象时，才会有值。此值为对象 ID

5.1.3.5.ACT_HI_PROCINST（历史流程实例表）

历史流程实例表。

表结构说明

字段名称 描述 数据类型 Nullable 取值说明
ID_ 主键 varchar(64) NO 主键 ID
PROC_INST_ID_ 流程实例 ID 键 varchar(64) NO 流程实例 ID 键
BUSINESS_KEY_ 业务主键 varchar(255) YES 业务主键，业务表单的 ID
PROC_DEF_ID_ 流程定义 ID varchar(64) NO 流程定义 ID
START_TIME_ 开始时间 datetime(3) NO 开始时间
END_TIME_ 结束时间 datetime(3) YES 结束时间
DURATION_ 耗时 bigint(20) YES 耗时
START_USER_ID_ 起草人 varchar(255) YES 起草人
START_ACT_ID_ 开始节点 ID varchar(255) YES 开始环节 ID
END_ACT_ID_ 结束节点 ID varchar(255) YES 结束环节 ID
SUPER_PROCESS_INSTANCE_ID_ 父流程实例 ID varchar(64) YES 父流程实例 ID
DELETE_REASON_ 删除原因 varchar(4000) YES 删除原因
TENANT_ID_ 租户 ID varchar(255) YES 租户 ID
NAME_ 名称 varchar(255) YES 名称

5.1.3.6.ACT_HI_IDENTITYLINK（历史流程人员表）

任务参与者数据表，主要存储历史节点参与者的信息。

表结构说明

字段名称 描述 数据类型 Nullable 取值说明
ID_ 主键 varchar(64) NO 主键 ID
GROUP_ID_ 用户组 ID varchar(255) YES 用户组 ID
TYPE_ 类型 varchar(255) YES assignee、candidate、owner、starter、participant
USER_ID_ 用户 ID varchar(255) YES 用户 ID
TASK_ID_ 节点实例 ID varchar(64) YES 节点实例 ID
PROC_INST_ID_ 流程实例 ID varchar(64) YES 流程实例 ID

5.1.3.7. ACT_HI_TASKINST（历史流程任务表）

表结构说明

字段名称 描述 数据类型 Nullable 取值说明
ID_ 主键 varchar(64) NO 主键 ID
PROC_DEF_ID_ 流程定义 ID varchar(64) YES 流程定义 ID
TASK_DEF_KEY_ 节点定义 ID varchar(255) YES 节点定义 ID
PROC_INST_ID_ 流程实例 ID varchar(64) YES 流程实例 ID
EXECUTION_ID_ 执行实例 ID varchar(64) YES 执行实例 ID
NAME_ 名称 varchar(255) YES 名称
PARENT_TASK_ID_ 父节点实例 ID varchar(64) YES 父节点实例 ID
DESCRIPTION_ 描述 varchar(4000) YES 描述
OWNER_ 实际签收人 任务的拥有者 varchar(255) YES 签收人（默认为空，只有在委托时才有值）
ASSIGNEE_ 签收人或被委托 varchar(255) YES 签收人或被委托
START_TIME_ 开始时间 datetime(3) NO 开始时间
CLAIM_TIME_ 提醒时间 datetime(3) YES 提醒时间
END_TIME_ 结束时间 datetime(3) YES 结束时间
DURATION_ 耗时 datetime(3) YES 耗时
DELETE_REASON_ 删除原因 varchar(4000) YES completed、deleted
PRIORITY_ 优先级别 int(11) YES 优先级别
DUE_DATE_ 过期时间 datetime(3) YES 过期时间，表明任务应在多长时间内完成
FORM_KEY_ 节点定义的 formkey varchar(255) YES desinger 节点定义的 form_key 属性
CATEGORY_ 类别 varchar(255) YES 类别
TENANT_ID_ 租户 ID varchar(255) YES 租户 ID

5.1.3.8. ACT_HI_VARINST（流程历史变量表）

流程历史变量表

表结构说明

字段名称 描述 数据类型 Nullable 取值说明

ID_ 主键 varchar(64) NO 主键 ID
PROC_INST_ID_ 流程实例 ID varchar(64) YES 流程实例 ID
EXECUTION_ID_ 执行实例 ID varchar(64) YES 执行实例 ID
TASK_ID_ 节点实例 ID varchar(64) YES 节点实例 ID
NAME_ 名称 varchar(255) NO 名称
VAR_TYPE_ 参数类型 varchar(100) YES jpa-entity、boolean、bytes、serializable、自定义 type(根据你自身配置)、CustomVariableType、date、double、integer、jpa-entity、long、null、short、string
REV_ 乐观锁 int(11) YES 默认值 NULL，version 版本
BYTEARRAY_ID_ 字节表 ID varchar(64) YES ACT_GE_BYTEARRAY 表的主键
DOUBLE_ double double YES 存储 DoubleType 类型的数据
LONG_ long bigint(20) YES 存储 LongType 类型的数据
TEXT_ 文本 varchar(4000) YES 存储变量值类型为 String，如此处存储持久化对象时，值 jpa 对象的 class
TEXT2_ 文本 varchar(4000) YES 此处存储的是 JPA 持久化对象时，才会有值。此值为对象 ID
CREATE_TIME_ 创建时间 datetime(3) YES 创建时间
LAST_UPDATED_TIME_ 最新更改时间 datetime(3) YES 最新更改时间

5.1.3.9. ACT_RU_EVENT_SUBSCR（监听信息表）

throwEvent、catchEvent 时间监听信息表。

表结构说明

字段名称 描述 数据类型 Nullable 取值说明
ID_ 主键 varchar(64) NO 主键 ID
REV_ 乐观锁 int(11) YES 默认值 NULL，version 版本
EVENT_TYPE_ 类型 varchar(255) NO 事件类型
EVENT_NAME_ 名称 varchar(255) YES 事件名称
EXECUTION_ID_ 执行实例 ID varchar(64) YES 执行实例 ID
PROC_INST_ID_ 流程实例 ID varchar(64) YES 流程实例 ID
ACTIVITY_ID_ 活动实例 ID varchar(64) YES 活动实例 ID
CONFIGURATION_ 配置 varchar(255) YES 流程定义的 Namespace 就是类别
CREATED_ 是否创建 timestamp(3) NO 默认值，当前系统时间戳（CURRENT_TIMESTAMP(3)）
PROC_DEF_ID_ 流程定义 ID varchar(64) YES 流程定义 ID
TENANT_ID_ 租户 ID varchar(255) YES 租户 ID

5.1.3.10. ACT_RU_EXECUTION（运行时流程执行实例表）

流程执行记录表。

表结构说明

字段名称 描述 数据类型 Nullable 取值说明
ID_ 主键 varchar(64) NO 主键 ID
REV_ 乐观锁 int(11) YES 默认值 NULL，version 版本
PROC_INST_ID_ 流程实例 ID varchar(64) YES 流程实例 ID
BUSINESS_KEY_ 业务主键 ID varchar(255) YES 业务主键 ID
PARENT_ID_ 父节点实例 ID varchar(64) YES 父节点实例 ID
PROC_DEF_ID_ 流程定义 ID varchar(64) YES 流程定义 ID
SUPER_EXEC_ varchar(64) YES
ACT_ID_ 节点实例 ID varchar(255) YES 节点实例 ID 即 ACT_HI_ACTINST 中 ID
IS_ACTIVE_ 是否存活 tinyint(4) YES 是否存活
IS_CONCURRENT_ 是否并行 tinyint(4) YES 是否为并行(true/false)
IS_SCOPE_ tinyint(4) YES
IS_EVENT_SCOPE_ tinyint(4) YES
SUSPENSION_STATE_ 是否挂起 int(11) YES 挂起状态（1：激活、2：挂起）
CACHED_ENT_STATE_ int(11) YES
TENANT_ID_ 租户 ID varchar(255) YES 租户 ID
NAME_ 名称 varchar(255) YES 名称
LOCK_TIME_ timestamp(3) YES

5.1.3.11. ACT_RU_IDENTITYLINK（运行时流程人员表，主要存储任务节点与参与者相关信息）

任务参与者数据表，主要存储当前节点参与者的信息。

表结构说明

字段名称 描述 数据类型 Nullable 取值说明
ID_ 主键 varchar(64) NO 主键 ID
REV_ 乐观锁 int(11) YES 默认值 NULL，version 版本
GROUP_ID_ 用户组 ID varchar(255) YES 用户组 ID
TYPE_ 类型 varchar(255) YES assignee、candidate、owner、starter、participant
USER_ID_ 用户 ID varchar(255) YES 用户 ID
TASK_ID_ 节点实例 ID varchar(64) YES 节点实例 ID
PROC_INST_ID_ 流程实例 ID varchar(64) YES 流程实例 ID
PROC_DEF_ID_ 流程定义 ID varchar(64) YES 流程定义 ID

5.1.3.12. ACT_RU_JOB（运行时定时任务数据表）

运行时定时任务数据表。

表结构说明

字段名称 描述 数据类型 Nullable 取值说明
ID_ 主键 varchar(64) NO 主键 ID
REV_ 乐观锁 int(11) YES 默认值 NULL，version 版本
TYPE_ 类型 varchar(255) NO 类型
LOCK_EXP_TIME_ 锁定释放时间 timestamp(3) YES 锁定释放时间
LOCK_OWNER_ 挂起者 varchar(255) YES 挂起者
EXCLUSIVE_ tinyint(1) YES
EXECUTION_ID_ 执行实例 ID varchar(64) YES 执行实例 ID
PROCESS_INSTANCE_ID_ 流程实例 ID varchar(64) YES 流程实例 ID
PROC_DEF_ID_ 流程定义 ID varchar(64) YES 流程定义 ID
RETRIES_ int(11) YES
EXCEPTION_STACK_ID_ 异常信息 ID varchar(64) YES 异常信息 ID
EXCEPTION_MSG_异常信息 varchar(4000) YES 异常信息
DUEDATE_ 到期时间 timestamp(3) YES 到期时间
REPEAT_ 重复 varchar(255) YES 重复
HANDLER_TYPE_ 处理类型 varchar(255) YES 处理类型
HANDLER_CFG_ 处理标识 varchar(4000) YES 处理标识
TENANT_ID_ 租户 ID varchar(255) YES 处理

5.1.3.13. ACT_RU_TASK（运行时任务节点表）

行时任务数据表。

表结构说明

字段名称 描述 数据类型 Nullable 取值说明
ID_ 主键 varchar(64) NO 主键 ID
REV_ 乐观锁 int(11) YES 默认值 NULL，version 版本
EXECUTION_ID_ 执行实例 ID varchar(64) YES 执行实例 ID
PROC_INST_ID_ 流程实例 ID varchar(64) YES 流程实例 ID
PROC_DEF_ID_ 流程定义 ID varchar(64) YES 流程定义 ID
NAME_ 节点定义名称 varchar(255) YES 节点定义名称
PARENT_TASK_ID_ 父节点实例 ID varchar(64) YES 父节点实例 ID
DESCRIPTION_ 描述 varchar(4000) YES 节点定义描述
TASK_DEF_KEY_ 节点定义的 KEY varchar(255) YES 任务定义的 ID
OWNER_实际签收人 varchar(255) YES 拥有者（一般情况下为空，只有在委托时才有值）
ASSIGNEE_ 签收人或委托人 varchar(255) YES 签收人或委托人
DELEGATION_ 委托类型 varchar(64) YES DelegationState 分为两种：PENDING，RESOLVED，如无委托则为空。
PRIORITY_ 优先级别 int(11) YES 优先级别，默认为：50
CREATE_TIME_ 创建时间 timestamp(3) YES 创建时间

DUE_DATE_ 过期时间 datetime(3) YES 过期时间
CATEGORY_ 类别 varchar(255) YES 类别
SUSPENSION_STATE_ 是否挂起 int(11) YES （1：代表激活、2：代表挂起）
TENANT_ID_ 租户 ID varchar(255) YES 租户 ID
FORM_KEY_ 节点定义的 formkey varchar(255) YES 表单 KEY

5.1.3.14. ACT_RU_VARIABLE（运行时流程变量数据表）

运行时流程变量数据表。

表结构说明

字段名称 描述 数据类型 Nullable 取值说明
ID_ 主键 varchar(64) NO 主键 ID
REV_ 乐观锁 int(11) YES 默认值 NULL，version 版本
TYPE_ 类型 varchar(255) NO jpa-entity、boolean、bytes、serializable、自定义 type(根据你自身配置)、CustomVariableType、date、double、integer、jpa-entity、long、null、short、string
NAME_ 名称 varchar(255) NO 变量名称
EXECUTION_ID_ 执行实例 ID varchar(64) YES 执行实例 ID
PROC_INST_ID_ 流程实例 ID varchar(64) YES 流程实例 ID
TASK_ID_ 节点实例 ID varchar(64) YES 节点实例 ID（Local）
BYTEARRAY_ID_ 字节表 ID varchar(64) YES ACT_GE_BYTEARRAY 的 ID_
DOUBLE_ Double double YES 存储变量类型为 Double
LONG_ Long bigint(20) YES 存储变量类型为 Long
TEXT_ Text varchar(4000) YES ‘存储变量值类型为 String,如此处存储持久化对象时，值 jpa 对象的 class
TEXT2_ Text varchar(4000) YES 此处存储的是 JPA 持久化对象时，才会有值。此值为对象 ID

5.2. 克隆入口函数

```
/**
 * 对给定流程实例进行克隆
 *
 * @author xiaohelong
 * @version 2017-11-07
 *  * email:xiaohelong2005@126.com
 *          xiaohelong2005@gmail.com
 *          twitter.com/xiaohelong
 */
@Service
@Transactional(readOnly = true)
public class ActCloneService extends BaseService {
```

```
    //Act History
    @Autowired
    private ActHiActinstService actHiActinstService;
    @Autowired
    private ActHiAttachmentService actHiAttachmentService;
    @Autowired
    private ActHiCommentService actHiCommentService;
    @Autowired
    private ActHiDetailService actHiDetailService;
    @Autowired
    private ActHiIdentitylinkService actHiIdentitylinkService;
    @Autowired
    private ActHiProcinstService actHiProcinstService;
    @Autowired
    private ActHiTaskinstService actHiTaskinstService;
    @Autowired
    private ActHiVarinstService actHiVarinstService;
    //Act Runtime
    @Autowired
    private ActRuEventSubscrService actRuEventSubscrService;
    @Autowired
    private ActRuExecutionService actRuExecutionService;
    @Autowired
    private ActRuIdentitylinkService actRuIdentitylinkService;
    @Autowired
    private ActRuJobService actRuJobService;
    @Autowired
    private ActRuTaskService actRuTaskService;
    @Autowired
    private ActRuVariableService actRuVariableService;

    /**
     * idSet
     * id 集合，旧 ID 为 Key,新 ID 为 Value,每得到一个不同的老 KD，
     * 就检测是否存在，存在即忽略，如果是检测到一个还没有加入的旧 ID，则加入，并且给出对应的新 ID
     */
    private Map<String,Map<String, String>> idSet=new HashMap<String, Map<String, String>>();//第一个 String 代表流程 ID,主要用于支持递归处理
    /**
     * 排除不需要改的的字段,特别有如 proc_def_id_字段
     */
    private Set<String> excludeFieldsSet=new HashSet<String>();
```

```
    /**
     * 根据指定的流程实例 ID,以及要克隆的份数，决定克隆多少份流程实例 ID。以指定的流程实例为根节点，对流程进行递归处理。
     * #act_hi_procinst 中 SUPER_PROCESS_INSTANCE_ID 存储了父级流程实例 ID
     * select * from act_hi_procinst where PROC_INST_ID_='7525af84eb34484dacfb67895b376734'
     * #ru
     * #act_ru_execution 中 Parent_ID 存储了父级 exeuction Id(并行线路，call 子流程时也存储父级实例 ID)
     * #act_ru_execution 中 Super_Exec_存储父流程(子流程时，存储父流程的在 act_ru_execution 中的 ID，而不是实例 ID，这是关联的一种方式)
     *
     * @param procInsID 欲复制的流程实例 ID
     * @param copies 欲复制的流程实例 ID
     * @return 克隆后的流程实例 ID 列表
     */
    @Transactional(readOnly = false)
    public List<ActHiProcinst> cloneProcInsByID(String procInsID, Integer copies) throws NoSuchMethodException, InstantiationException, IllegalAccessException, InvocationTargetException {
        List<ActHiProcinst> retVal = new ArrayList<ActHiProcinst>();
        ActRuExecution actRuExecution = actRuExecutionService.get(procInsID);
        //here need to check procInsID to assure it is a  top level process instance
        if (actRuExecution == null) {
            //not running
            logger.info(procInsID + " process Instance is not running");//if you want to know more ,throw self defined exection here
            //Throw a running execption
            return null;
        }
        if (!actRuExecutionService.isTopLevel(actRuExecution)) {
            logger.info(actRuExecution.getId() + " process instance is not top level instance return");//if you want to know more ,throw self defined exection here
            return null;
        }
        if (copies > 0) {
            for (int i = 0; i < copies; i++) {
                ActHiProcinst newClonedProcinst = cloneProcInsByIDOnlyOneCopy(procInsID,null);
                if (newClonedProcinst != null) {
                    logger.info(" newId " + newClonedProcinst.getId() + " cloned");
                    retVal.add(newClonedProcinst);
                }
            }
```

```
        }
        return retVal;
    }

    /**
     * 核心思路：将此流程实例相关的的所有信息进行收集，再利用全文替换的思想，将相关联的 ID 号统一替换成新的 ID 号，这样就可以保证复制的流程关系一模一样。
     * 这里主要全文替换是 ID，且是几个特有的字段。
     *
     * @param procInstanceID 需要克隆的实例流程
     * @param newParentId 新的父亲的 ID,主要用于对子流程的递归
     * @return
     * @throws InvocationTargetException
     * @throws NoSuchMethodException
     * @throws InstantiationException
     * @throws IllegalAccessException
     */
    @Transactional(readOnly = false)
    public ActHiProcinst cloneProcInsByIDOnlyOneCopy(String procInstanceID,String newParentId) {
        ActHiProcinst needCloneProc=actHiProcinstService.get(procInstanceID);
        if(needCloneProc==null){//can't get the data
            return null;
        }
        initData(procInstanceID);
        Map<String,String> subIdSet=idSet.get(procInstanceID);

        if(!StringUtils.isBlank(newParentId))
        {//subprocess and it is in recursive call,need to set the new parentid to replace old parentid
            subIdSet.put(needCloneProc.getSuperProcessInstanceId(),newParentId);//this is the relation created;
        }
        //1. act_hi_actinst table
        ActHiActinst actHiActinstFindEntity = new ActHiActinst();
        actHiActinstFindEntity.setProcInstId(procInstanceID);
        List<ActHiActinst> actHiActinsts = actHiActinstService.findList(actHiActinstFindEntity);
        if(actHiActinsts!=null&&actHiActinsts.size()>0) {
            ActIdReplace<ActHiActinst> actHiActinstActIdReplace = new ActIdReplace<ActHiActinst>(ActHiActinst.class);
            actHiActinstActIdReplace.replaceCollection(actHiActinsts, subIdSet, excludeFieldsSet);//传过去的值按 java 的引用传递规则会对应更改并返回
            actHiActinstService.saveBatch(actHiActinsts);
```

```
        }

        //actHiActinstService.saveBatch(actHiActinsts);
        //2. act_hi_attachment table
        ActHiAttachment actHiAttachmentFindEntity = new ActHiAttachment();
        actHiAttachmentFindEntity.setProcInstId(procInstanceID);
        List<ActHiAttachment> actHiAttachments = actHiAttachmentService.findList(actHiAttachmentFindEntity);
        if(actHiAttachments!=null&&actHiAttachments.size()>0) {
            ActIdReplace<ActHiAttachment> actHiAttachmentActIdReplace = new ActIdReplace<ActHiAttachment>(ActHiAttachment.class);
            actHiAttachmentActIdReplace.replaceCollection(actHiAttachments, subIdSet, excludeFieldsSet);//传过去的值按 java 的引用传递规则会对应更改并返回
            actHiAttachmentService.saveBatch(actHiAttachments);
        }
        //3. act_hi_comment table
        //在系统中驳回和签收中没有写入流程实例（有为 3rf 空的，也有不为空的，需要进行特殊处理即找出所有流程相关的任务或者流程本身再去重）
        //可以先通过 taskinst 找到所有任务，再进行直接通过 ProcInstId 找出的记录，去重即可。
        ActHiComment actHiCommentFindEntity = new ActHiComment();
        actHiCommentFindEntity.setProcInstId(procInstanceID);
        List<ActHiComment> actHiComments = actHiCommentService.findAllCommentByProcInstId(actHiCommentFindEntity);
        if(actHiComments!=null&&actHiComments.size()>0) {
            ActIdReplace<ActHiComment> actHiCommentActIdReplace = new ActIdReplace<ActHiComment>(ActHiComment.class);
            actHiCommentActIdReplace.replaceCollection(actHiComments, subIdSet, excludeFieldsSet);//传过去的值按 java 的引用传递规则会对应更改并返回
            actHiCommentService.saveBatch(actHiComments);
        }
        //4. act_hi_detail table
        ActHiDetail actHiDetailFindEntity = new ActHiDetail();
        actHiDetailFindEntity.setProcInstId(procInstanceID);
        List<ActHiDetail> actHiDetails = actHiDetailService.findList(actHiDetailFindEntity);
        if(actHiDetails!=null&&actHiDetails.size()>0) {
            ActIdReplace<ActHiDetail> actHiDetailActIdReplace = new ActIdReplace<ActHiDetail>(ActHiDetail.class);
            actHiDetailActIdReplace.replaceCollection(actHiDetails, subIdSet, excludeFieldsSet);//传过去的值按 java 的引用传递规则会对应更改并返回
            actHiDetailService.saveBatch(actHiDetails);
        }
        //5. act_hi_identitylink table
        //身份关链表（有为空的，也有不为空的，需要进行特殊处理即找出所有流程相关
```

```
的任务或者流程本身再去重）
        //可以先通过 taskinst 找到所有任务，再进行直接通过 ProcInstId 找出的记录，去重即可。
        ActHiIdentitylink actHiIdentitylinkFindEntity = new ActHiIdentitylink();
        actHiIdentitylinkFindEntity.setProcInstId(procInstanceID);
        List<ActHiIdentitylink> actHiIdentitylinks = actHiIdentitylinkService.findAllIdentitylinkByProcInstId(actHiIdentitylinkFindEntity);
        if(actHiIdentitylinks!=null&&actHiIdentitylinks.size()>0) {
            ActIdReplace<ActHiIdentitylink> actHiIdentitylinkActIdReplace = new ActIdReplace<ActHiIdentitylink>(ActHiIdentitylink.class);
            actHiIdentitylinkActIdReplace.replaceCollection(actHiIdentitylinks, subIdSet, excludeFieldsSet);//传过去的值按 java 的引用传递规则会对应更改并返回
            actHiIdentitylinkService.saveBatch(actHiIdentitylinks);
        }
        //6.act_hi_procinst table
        ActHiProcinst actHiProcinstFindEntity = new ActHiProcinst();
        actHiProcinstFindEntity.setProcInstId(procInstanceID);
        List<ActHiProcinst> actHiProcinsts = actHiProcinstService.findList(actHiProcinstFindEntity);
        if(actHiProcinsts!=null&&actHiProcinsts.size()>0)
        {
            ActIdReplace<ActHiProcinst> actHiProcinstActIdReplace=new ActIdReplace<ActHiProcinst>(ActHiProcinst.class);

            actHiProcinstActIdReplace.replaceCollection(actHiProcinsts,subIdSet,excludeFieldsSet);//传过去的值按 java 的引用传递规则会对应更改并返回
            actHiProcinstService.saveBatch(actHiProcinsts);
        }
        //7.act_hi_taskinst table
        ActHiTaskinst actHiTaskinstFindEntity = new ActHiTaskinst();
        actHiTaskinstFindEntity.setProcInstId(procInstanceID);
        List<ActHiTaskinst> actHiTaskinsts = actHiTaskinstService.findList(actHiTaskinstFindEntity);
        if(actHiTaskinsts!=null&&actHiActinsts.size()>0) {
            ActIdReplace<ActHiTaskinst> actHiTaskinstActIdReplace = new ActIdReplace<ActHiTaskinst>(ActHiTaskinst.class);
            actHiTaskinstActIdReplace.replaceCollection(actHiTaskinsts, subIdSet, excludeFieldsSet);//传过去的值按 java 的引用传递规则会对应更改并返回
            actHiTaskinstService.saveBatch(actHiTaskinsts);
        }
        //8.act_hi_varinst
        ActHiVarinst actHiVarinstFindEntity = new ActHiVarinst();
        actHiVarinstFindEntity.setProcInstId(procInstanceID);
        List<ActHiVarinst> actHiVarinsts =
```

```
actHiVarinstService.findList(actHiVarinstFindEntity);
        if(actHiVarinsts!=null&&actHiVarinsts.size()>0) {
            ActIdReplace<ActHiVarinst> actHiVarinstActIdReplace = new
ActIdReplace<ActHiVarinst>(ActHiVarinst.class);
            actHiVarinstActIdReplace.replaceCollection(actHiVarinsts, subIdSet,
excludeFieldsSet);//传过去的值按 java 的引用传递规则会对应更改并返回
            actHiVarinstService.saveBatch(actHiVarinsts);
        }
        //act_RU part
        //1.act_ru_event_subscr table
        ActRuEventSubscr actRuEventSubscrFindEntity = new ActRuEventSubscr();
        actRuEventSubscrFindEntity.setProcInstId(procInstanceID);
        List<ActRuEventSubscr> actRuEventSubscrs =
actRuEventSubscrService.findList(actRuEventSubscrFindEntity);
        if(actRuEventSubscrs!=null&&actRuEventSubscrs.size()>0) {
            ActIdReplace<ActRuEventSubscr> actRuEventSubscrActIdReplace = new
ActIdReplace<ActRuEventSubscr>(ActRuEventSubscr.class);
            actRuEventSubscrActIdReplace.replaceCollection(actRuEventSubscrs, subIdSet,
excludeFieldsSet);//传过去的值按 java 的引用传递规则会对应更改并返回
            actRuEventSubscrService.saveBatch(actRuEventSubscrs);
        }
        //2.act_ru_execution
        ActRuExecution actRuExecutionFindEntity = new ActRuExecution();
        actRuExecutionFindEntity.setProcInstId(procInstanceID);
        List<ActRuExecution> actRuExecutions =
actRuExecutionService.findList(actRuExecutionFindEntity);
        if(actRuExecutions!=null&&actRuExecutions.size()>0)
        {
            ActIdReplace<ActRuExecution> actRuExecutionActIdReplace=new
ActIdReplace<ActRuExecution>(ActRuExecution.class);

actRuExecutionActIdReplace.replaceCollection(actRuExecutions,subIdSet,excludeFieldsSet);//
传过去的值按 java 的引用传递规则会对应更改并返回
            actRuExecutionService.saveBatch(actRuExecutions);
        }
        //身份关链表（有为空的，也有不为空的，需要进行特殊处理即找出所有流程相关
的任务或者流程本身再去重）

        //3.act_ru_identityservice table
        ActRuIdentitylink actRuIdentitylinkFindEntity = new ActRuIdentitylink();
        actRuIdentitylinkFindEntity.setProcInstId(procInstanceID);
        List<ActRuIdentitylink> actRuIdentitylinks =
actRuIdentitylinkService.findAllIdentitylinkByProcInstId(actRuIdentitylinkFindEntity);
        if(actRuIdentitylinks!=null&&actRuIdentitylinks.size()>0)
```

```
        {
            ActIdReplace<ActRuIdentitylink> actRuIdentitylinkActIdReplace=new
ActIdReplace<ActRuIdentitylink>(ActRuIdentitylink.class);

actRuIdentitylinkActIdReplace.replaceCollection(actRuIdentitylinks,subIdSet,excludeFieldsSet)
;//传过去的值按 java 的引用传递规则会对应更改并返回
            actRuIdentitylinkService.saveBatch(actRuIdentitylinks);
        }

        //5.act_ru_task table
        ActRuTask actRuTaskFindEntity = new ActRuTask();
        actRuTaskFindEntity.setProcInstId(procInstanceID);
        List<ActRuTask> actRuTasks = actRuTaskService.findList(actRuTaskFindEntity);
        if(actRuTasks!=null&&actRuTasks.size()>0)
        {
            ActIdReplace<ActRuTask> actRuTaskActIdReplace=new
ActIdReplace<ActRuTask>(ActRuTask.class);

actRuTaskActIdReplace.replaceCollection(actRuTasks,subIdSet,excludeFieldsSet);//传过去的
值按 java 的引用传递规则会对应更改并返回
            actRuTaskService.saveBatch(actRuTasks);
        }

        //4.act_ru_job table
        ActRuJob actRuJobFindEnity = new ActRuJob();
        actRuJobFindEnity.setProcessInstanceId(procInstanceID);
        List<ActRuJob> actRuJobs = actRuJobService.findList(actRuJobFindEnity);
        if(actRuJobs!=null&&actRuJobs.size()>0) {
            ActIdReplace<ActRuJob> actRuJobActIdReplace = new
ActIdReplace<ActRuJob>(ActRuJob.class);
            actRuJobActIdReplace.replaceCollection(actRuJobs, subIdSet,
excludeFieldsSet);//传过去的值按 java 的引用传递规则会对应更改并返回
            actRuJobService.saveBatch(actRuJobs);
        }

        //6.act_ru_variable table
        ActRuVariable actRuVariableFindEntity = new ActRuVariable();
        actRuVariableFindEntity.setProcInstId(procInstanceID);
        List<ActRuVariable> actRuVariables =
actRuVariableService.findList(actRuVariableFindEntity);
        if(actRuVariables!=null&&actRuVariables.size()>0) {
            ActIdReplace<ActRuVariable> actRuVariableActIdReplace = new
ActIdReplace<ActRuVariable>(ActRuVariable.class);
            actRuVariableActIdReplace.replaceCollection(actRuVariables, subIdSet,
```

```
excludeFieldsSet);//传过去的值按 java 的引用传递规则会对应更改并返回
                actRuVariableService.saveBatch(actRuVariables);
            }
            //对于正在执行的也会在 hi_proc_inst 中有记录，因此只要从 hi_proc_inst 中获取到子流程 ID 即可进行递归处理。
            ActHiProcinst findChildProc=new ActHiProcinst();
            findChildProc.setSuperProcessInstanceId(procInstanceID);
            List<ActHiProcinst>
childProcesses=actHiProcinstService.findChildActHiProcinst(findChildProc);
            if(childProcesses!=null){
                for(int i=0;i<childProcesses.size();i++){
                    //recursive call  递归时，是需要以已完成的流程实例为基础,，因为递归时需要复制已经结束的(正在运行的肯定要支持),并且需要父节点更新过去。
                    ActHiProcinst childProcess=childProcesses.get(i);

cloneProcInsByIDOnlyOneCopy(childProcess.getId(),subIdSet.get(procInstanceID));// 子 递 归时，需要将新的父 ID 传入,不需要保存该返回值，因为直接存入了数据库，只需要在顶级获取即可
                }
            }
            ActHiProcinst newCloneProc=null;
            try {

                newCloneProc=(ActHiProcinst) BeanUtils.cloneBean(needCloneProc);
                newCloneProc.setId(subIdSet.get(procInstanceID));//返回新的 ID 值
                newCloneProc.setProcInstId(subIdSet.get(procInstanceID));//返回新的 ID 值实例
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            } catch (InstantiationException e) {
                e.printStackTrace();
            } catch (InvocationTargetException e) {
                e.printStackTrace();
            } catch (NoSuchMethodException e) {
                e.printStackTrace();
            }
            idSet.put(procInstanceID,subIdSet);//对更改过的值进行回更新
            return newCloneProc;//todo  返回新克隆的 ID
        }
        /**
         * Constructor  初始化
         */
        private void initData(String procInstId) {
            Map<String,String> subIdSet=new HashMap<String, String>();
```

```
            idSet.put(procInstId,subIdSet);
            subIdSet.put(procInstId,IdGen.uuid());//initialize
            excludeFieldsSet.add("getProcDefId");
            excludeFieldsSet.add("getBytearrayId");
        }
}
```

5.3. 最核心的函数

```
/**
 * 针对 act 需要克隆的类别，做一个模板通用工具，免得每一个都要写一次
 * Created by xiaohelong on 2017/11/11.
 *   * email:xiaohelong2005@126.com
 *           xiaohelong2005@gmail.com
 *           twitter.com/xiaohelong
 */
public class ActIdReplace<T extends DataEntity<T>> {
    protected Logger logger = LoggerFactory.getLogger(getClass());
    private final Class<T> clazz;
    public ActIdReplace(Class<T> clazz){
        this.clazz=clazz;
    }
    /**
     * 替换指定集合的 ID，这里利用了 Java 函数的参数值传递为引用传递的特性，即会修改传过来的参数（如果没有，那就是错的，这里必须是引用规则)
     * @param dataList
     * @param idSet
     * @param excludeFieldsSet
     */
    public void replaceCollection(List<T> dataList, Map<String, String> idSet,  Set<String> excludeFieldsSet){
        if (dataList != null && dataList.size() > 0) {
            try {
                for (int i = 0; i < dataList.size(); i++) {
                    T data = dataList.get(i);
                    data.setIsNewRecord(true);
                    /**循环读取所有属性值(核心代码区 kernel code)
                     * 1.检测是不是排名和属性名
                     * 2.并检使用该值检测对应的值是不是字符型
                     * 3.如果是字符，则是否符合 uuid 的正则'
                     * 4.如果是 id，则看此 id 是否已在 id 集合中，则忽略，如果集合中没有出现过，则加入该集合。
                     * 5.为了节省效率，直接在此循环中，将实体的 id 替换为集合中旧实体对应的 id(不要再在后面统一处理了，减少循环，就会提升效率）
```

```
                */
                Method[] publicMethods = Class.forName(clazz.getName()).getMethods();//get all String fields(private protected public
                if (publicMethods != null && publicMethods.length > 0) {
                    for (int m = 0; m < publicMethods.length; m++) {
                        String methodName = publicMethods[m].getName();
                        if ((!excludeFieldsSet.contains(methodName) && methodName.toLowerCase().contains("id")&&methodName.startsWith("get"))||methodName.equals("getSuperExec")) {//notExcluded and its' name start with get
                            Class returnType=publicMethods[m].getReturnType();//获取返回值的类型
                            if(returnType.getName().equals(String.class.getName())){//id 只有字符串里面才会有的
                                logger.info("get method name:"+methodName+" in class"+data.getClass().getName());
                                String getFieldData = (String) publicMethods[m].invoke(data);
                                //act 里面的 id 都是 32 位的,用 DbIdGenerator 生成,为了以示区别，我们用 IdGen 生成的 32 位的。生成

                                if(getFieldData!=null&&getFieldData.length()==32)
                                {
                                    if(!idSet.containsKey(getFieldData)){
                                        idSet.put(getFieldData, IdGen.uuid());
                                    }
                                    //调用 set 函数，传入新 ID 值
                                    String newId=idSet.get(getFieldData);// 找出对应的新的用于替换的 ID 值，调用 set 命令
                                    String setMethodName=methodName.replace("get","set");
                                    Method setMethod=data.getClass().getMethod(setMethodName,String.class);
                                    logger.info("set method name:"+setMethodName+" in class"+data.getClass().getName());
                                    setMethod.invoke(data,newId);//将老的 ID 改至对应的新 ID，全部
                                }
                            }
                        }
                    }
                } catch (ClassNotFoundException e) {
                    logger.error(e.getMessage());
```

```
                } catch (InvocationTargetException e) {
                    logger.error(e.getMessage());
                } catch (IllegalAccessException e) {
                    logger.error(e.getMessage());
                }catch (NoSuchMethodException e){
;                    logger.error(e.getMessage());
                }
            }
        }
}
```

5.4. 其它应该注意的事项

5.4.1. 读取 act_hi_comment table

```
    //在系统中驳回和签收中没有写入流程实例（有为 3rf 空的，也有不为空的，需要进行特殊处理即找出所有流程相关的任务或者流程本身）
    <select id="findAllCommentByProcInstId" resultType="ActHiComment">
        SELECT
        <include refid="actHiCommentColumns"/>
        FROM act_hi_comment a
        <include refid="actHiCommentJoins"/>
        join act_hi_taskinst t on a.task_id_=t.id_
        <where>
            <if test="procInstId != null">
                t.proc_inst_id_=#{procInstId}
            </if>
        </where>
        <choose>
            <when test="page !=null and page.orderBy != null and page.orderBy != ''">
                ORDER BY ${page.orderBy}
            </when>
            <otherwise>
            </otherwise>
        </choose>
    </select>
```

5.4.2. 读取 act_hi_identitylink table 时注意

```
    //身份关链表（有为空的，也有不为空的，需要进行特殊处理即找出所有流程相关的任务或者流程本身）
    <select id="findAllIdentitylinkByProcInstId" resultType="ActHiIdentitylink">
```

```
        SELECT
        <include refid="actHiIdentitylinkColumns"/>
        FROM act_hi_identitylink a
        <include refid="actHiIdentitylinkJoins"/>
        join act_hi_taskinst t on a.task_id_=t.id_
        <where>
            <if test="procInstId != null">
                t.proc_inst_id_=#{procInstId}
            </if>
        </where>
        <choose>
            <when test="page !=null and page.orderBy != null and page.orderBy != ''">
                ORDER BY ${page.orderBy}
            </when>
            <otherwise>
            </otherwise>
        </choose>
    </select>
```

5.4.3. 批量保存 act_ru 时，需要关闭外键检测 Close foreignkey check

```
<insert id="saveBatch">
        set FOREIGN_KEY_CHECKS=0;
        INSERT INTO act_ru_execution(
        id_,
        rev_,
        proc_inst_id_,
        business_key_,
        parent_id_,
        proc_def_id_,
        super_exec_,
        act_id_,
        is_active_,
        is_concurrent_,
        is_scope_,
        is_event_scope_,
        suspension_state_,
        cached_ent_state_,
        tenant_id_,
        name_,
        lock_time_
        ) VALUES
```

```
        <foreach item="item" index="index" collection="list" separator=",">
         (
        #{item.id},
        #{item.rev},
        #{item.procInstId},
        #{item.businessKey},
        #{item.parentId},
        #{item.procDefId},
        #{item.superExec},
        #{item.actId},
        #{item.isActive},
        #{item.isConcurrent},
        #{item.isScope},
        #{item.isEventScope},
        #{item.suspensionState},
        #{item.cachedEntState},
        #{item.tenantId},
        #{item.name},
        #{item.lockTime}
        )
        </foreach>;
        set FOREIGN_KEY_CHECKS=1;
    </insert>
```

5.5. 代码 Code

Github:https://github.com/xiaohelong/ActivitiOrFlowableProcessInstanceCloneSolution