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](mailto:xiaohelong2005@126.com) 2017-11-24 20:24 长沙

# 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。

# 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。

# 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。

# 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\_\*。
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。

# 最佳实践

## 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）

### 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

### 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

### 表结构说明(关键ID字段加粗,要包括本身的ID\_号)

#### 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``，

#### 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

#### 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

#### 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

#### 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

#### 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**

#### 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

#### 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

#### 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

#### 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

#### 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

#### 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

#### 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

#### 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

## 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");

}

}

## 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());

}

}

}

}

## 其它应该注意的事项

### 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>

### 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>

### 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>

## 代码Code

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