Flowable开发文档

# Flowable与spring boot整合；

（一）只需要在pom.xml配置好以下(加粗部分）：

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.zjm</groupId>

<artifactId>gwork</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>gwork</name>

<description>gwork project for Spring Boot</description>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.4.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

<flowable.version>6.4.1</flowable.version>

<mysql.version>5.1.46</mysql.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

<exclusions>

<exclusion>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-logging</artifactId>

</exclusion>

</exclusions>

</dependency>

<!--日志配置 -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-log4j2</artifactId>

</dependency>

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

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

<version>1.3.2</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-configuration-processor</artifactId>

<optional>true</optional>

</dependency>

<!--flowable set -->

<dependency>

<groupId>org.flowable</groupId>

<artifactId>flowable-spring-boot-starter</artifactId>

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

</dependency>

<dependency>

<groupId>mysql</groupId>

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

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

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<!--======以下依赖内容为zhongzk 增加 begin========== -->

<!-- spring security 和 jwt -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-core</artifactId>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.datatype</groupId>

<artifactId>jackson-datatype-joda</artifactId>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.module</groupId>

<artifactId>jackson-module-parameter-names</artifactId>

</dependency>

<!-- 分页插件 -->

<dependency>

<groupId>com.github.pagehelper</groupId>

<artifactId>pagehelper-spring-boot-starter</artifactId>

<version>1.2.5</version>

</dependency>

<!-- alibaba的druid 数据库连接池 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid-spring-boot-starter</artifactId>

<version>1.1.9</version>

</dependency>

<!-- javax.servlet-api -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<scope>provided</scope>

</dependency>

<!-- mockmaker 依赖 -->

<dependency>

<groupId>net.bytebuddy</groupId>

<artifactId>byte-buddy</artifactId>

</dependency>

<dependency>

<groupId>net.bytebuddy</groupId>

<artifactId>byte-buddy-agent</artifactId>

<scope>test</scope>

</dependency>

<!-- mybatis-generator-maven-plugin 插件2010.4.1 zhongzk -->

<dependency>

<groupId>org.mybatis.generator</groupId>

<artifactId>mybatis-generator-maven-plugin</artifactId>

<version>1.3.7</version>

</dependency>

<!-- poi 开源包zhongzk 2019.4.6-->

<dependency>

<groupId>org.apache.poi</groupId>

<artifactId>poi</artifactId>

<version>3.17</version>

</dependency>

<dependency>

<groupId>org.apache.poi</groupId>

<artifactId>poi-examples</artifactId>

<version>3.17</version>

</dependency>

<dependency>

<groupId>org.apache.poi</groupId>

<artifactId>poi-excelant</artifactId>

<version>3.17</version>

</dependency>

<dependency>

<groupId>org.apache.poi</groupId>

<artifactId> poi-scratchpad</artifactId>

<version>3.17</version>

</dependency>

<dependency>

<groupId>org.apache.poi</groupId>

<artifactId>poi-ooxml</artifactId>

<version>3.17</version>

</dependency>

<dependency>

<groupId>org.apache.poi</groupId>

<artifactId>poi-ooxml-schemas</artifactId>

<version>3.17</version>

</dependency>

<!-- common csv zhongzk 2019.4.6 -->

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-csv</artifactId>

<version>1.6</version>

</dependency>

<!--=====依赖内容为zhongzk 增加 end==== -->

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

<!-- zhongzk 2010.4.1 mybatis-generator插件-->

<plugin>

<groupId>org.mybatis.generator</groupId>

<artifactId>mybatis-generator-maven-plugin</artifactId>

<version>1.3.2</version>

<configuration>

<configurationFile>src/main/resources/generatorConfig.xml</configurationFile>

<verbose>true</verbose>

<overwrite>true</overwrite>

</configuration>

<executions>

<execution>

<id>Generate MyBatis Artifacts</id>

<goals>

<goal>generate</goal>

</goals>

</execution>

</executions>

<dependencies>

<dependency>

<groupId>org.mybatis.generator</groupId>

<artifactId>mybatis-generator-core</artifactId>

<version>1.3.2</version>

</dependency>

</dependencies>

</plugin>

</plugins>

<resources>

<resource>

<directory>src/main/java</directory>

<includes>

<include>\*\*/\*.xml</include>

</includes>

</resource>

</resources>

</build>

</project>

（二）在spring boot主main类中，关掉spring security认证以及配置Mapper接口的扫描：

package com.zjm.gwork;

import javax.servlet.MultipartConfigElement;

import org.mybatis.spring.annotation.MapperScan;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.web.servlet.MultipartConfigFactory;

import org.springframework.boot.web.servlet.ServletComponentScan;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import java.io.File;

@SpringBootApplication

//添加对mapper包扫描

@MapperScan("com.zjm.gwork.\*\*.mapper")

//过滤器开关

@ServletComponentScan("com.zjm.gwork.utils")

//取消spring security的认证

@EnableAutoConfiguration(exclude = {org.springframework.boot.autoconfigure.security.servlet.SecurityAutoConfiguration.class})

//开启缓存功能

//@EnableCaching

//启动swagger注解

public class GworkApplication {

public static void main(String[] args) {

SpringApplication.run(GworkApplication.class, args);

}

/\*\*

\* 文件上传配置 zhongzk

\* @return

\*/

@Bean

public MultipartConfigElement multipartConfigElement() {

MultipartConfigFactory factory = new MultipartConfigFactory();

//单个文件最大

factory.setMaxFileSize("300MB"); //KB,MB

// 设置总上传数据总大小

factory.setMaxRequestSize("500MB");

// 临时文件路径

String tempUrl = System.getProperty("user.dir") + File.separator + "bjgwork" + File.separator + "tmp";

System.out.println("临时目录：" + tempUrl);

File file = new File(tempUrl);

if (!file.exists()) {

file.mkdirs();

}

factory.setLocation(tempUrl);

return factory.createMultipartConfig();

}

}

（三）application.yml配置好数据库的连接池，如下：

server:

port: 7001

spring:

datasource:

#基本属性

url: jdbc:mysql://localhost:3306/gwork?useUnicode=true&characterEncoding=UTF-8&allowMultiQueries=true&useSSL=false

username: root

password: root

#druid相关配置

druid:

#监控统计拦截的filters

filters: stat

driver-class-name: com.mysql.jdbc.Driver

#配置初始化大小/最小/最大

initial-size: 2

min-idle: 2

max-active: 30

#获取连接等待超时时间

max-wait: 60000

#间隔多久进行一次检测，检测需要关闭的空闲连接

time-between-eviction-runs-millis: 60000

#一个连接在池中最小生存的时间

min-evictable-idle-time-millis: 300000

validation-query: SELECT 'x'

test-while-idle: true

test-on-borrow: false

test-on-return: false

#打开PSCache，并指定每个连接上PSCache的大小。oracle设为true，mysql设为false。分库分表较多推荐设置为false

pool-prepared-statements: false

max-pool-prepared-statement-per-connection-size: 20

#上传附件的目录设置以及系统一块模板word、excel的存放路径 zhongzk 2019.4.8

#这样做的原因是由于spring boot发布时打包成了jar,所以没有办法往jar中写文件

filepath:

uploadpath: c:\uploadfile\

templatepath: c:\templatefile\

mybatis:

#mapper-locations: classpath:mapper/\*.xml

mapper-locations: classpath\*:com/zjm/gwork/\*\*/mapper/\*mapper.xml

#type-aliases-package: com.zjm.gwork.model

type-aliases-package: com.zjm.gwork.\*\*.model

#pagehelper

pagehelper:

helperDialect: mysql

reasonable: true

supportMethodsArguments: true

params: count=countSql

returnPageInfo: check

#flowable 取消spring boot时自动部署resource/processes中的流程文件

flowable:

check-process-definitions: false

#db-identity-used: true

# 自动生成flowable相关表 第一次生成后建议关闭提高运行速度

database-schema-update: true

# 保存历史数据级别设置为full最高级别，便于历史数据的追溯

history-level: full

（四）启动spring boot后自动生成70张数据库表，建议安装JRebel：秒级热更新神器必备。

# flowable流程模型管理接口

一、流程模型资源

  流程定义文件：文件的命名必须是XXXXbpmn20.xml，注意命名规范，后面必须是bpmn20.xml。

  流程定义图片：用BPMN2.0规范定义的各种图形描绘（BMPN2.0的符号及画布上的坐标信息），一般是PNG格式。

  表单文件：把表单内容保存在一个文件中，扩展名为drl。

  规则文件：扩展名为drl。

二、流程模型资源的部署：

        用flowable Modeler设计好的流程图后，有多种资源部署方式，分别是（1）导出xml，然后部署到flowable数据库中 （2）多个流程模型定义文件，即xml打包成zip/bar，多个资源文件一起部署。（3）Model方式部署，设计器中的发布就是用的此方法。（4）其它方式部署

（1）流程资源xml部署：

/\*

\* @Description: 部署工作流程

\*

\*

\* @Date: 2019/4/27 1:53

\* @Param: [filePath, flowName, flowKey]

\* @Param: [filePath, flowName, flowKey]

\* @return: com.zjm.gwork.utils.Result

\*\*/

public Deployment deployFlow(String filePath) {

try {

DeploymentBuilder deploymentBuilder = repositoryService

.createDeployment()

.addClasspathResource(filePath);

Deployment deployment = deploymentBuilder.deploy();

logger.info("成功：部署工作流成：" + filePath);

return deployment;

} catch (Exception e) {

logger.error("失败：部署工作流：" + e);

return null;

} // end catch

}

（2）zip/bar打包，多个流程资源文件部署：

 InputStream in = this.getClass().getClassLoader().getResourceAsStream("diagrams/approve.zip");

ZipInputStream zipInputStream = new ZipInputStream(in);

 Deployment deployment = processEngine.getRepositoryService()//获取流程定义和部署对象相关的Service

                                         .createDeployment()//创建部署对象

                                         .addZipInputStream(zipInputStream)//使用zip方式部署，将approve.bpmn和approve.png压缩成zip格式的文件

                                         .deploy();//完成部署

 System.out.println("部署ID："+deployment.getId());

 System.out.println("部署时间："+deployment.getDeploymentTime());

(3)Model部署

  通过act\_de\_model中存放的Modeler内容来部署，例如代码（此代码别人写的，作者不详，但在此感谢此作者）：

org.flowable.ui.modeler.domain.Model modelData =modelService.getModel(modelId);

        //获取模型

        byte[] bytes = modelService.getBpmnXML(modelData);

        if (bytes == null) {

            return (new PermissionResult()).getPermissionResult(500,"模型数据为空，请先设计流程并成功保存，再进行发布。");

        }

        BpmnModel model=modelService.getBpmnModel(modelData);

        if(model.getProcesses().size()==0){

            return (new PermissionResult()).getPermissionResult(500,"数据模型不符要求，请至少设计一条主线流程。");

        }

        byte[] bpmnBytes = new BpmnXMLConverter().convertToXML(model);

        //发布流程

        String processName = modelData.getName() + ".bpmn20.xml";

   Deployment  deploy=   repositoryService.createDeployment()

                .name(modelData.getName())

                .addString(processName, new String(bpmnBytes, "UTF-8"))

                .deploy();

        return (new PermissionResult()).getPermissionResult(200,"部署成功",deploy);

(4)其它方式部署：

    你也可以用代码创建bpmnModel模型，然后用以下代码进行部署：           repositoryService.createDeployment().addBpmnModel("bpmnModel", bpmnModel).deploy();

三、部署有关的数据库表

act\_re\_deployment：流程模型部署对象表

    每部署一次生成一条记录，首先生成这条数据，它的id主键将会被act\_re\_procdef和act\_ge\_bytearray作为外键。

act\_re\_procdef（流程定义表）

   很多人不理解为什么还要这张表，需要这张表是因为有时候一次部署可能采用zip/bar进行部署，里面是有多份流程定义文件xml的，这时候act\_re\_deployment只有一条部署信息，但act\_re\_procdef有多个记录（一个流程定义对应一条），同时act\_ge\_bytearray也是每一个流程定义对应2条记录。这个表有DEPLOYMENT\_ID\_外键字段，用它关联act\_re\_deployment。

act\_ge\_bytearray（资源文件表）

    流程模型资源文件的真正存放地方，它每部署一次就会产生2条记录，一条是关于bpmn规范的文件内容存放在BYTES字段中，另一条是图片信息，采用二进制格式存储。提示：可以部署后解析bpmn文件的内容自动生成流程图，实现流程图的跟踪线路。

act\_ge\_property（主键生成表）：部署主键的id生成规则。

act\_re\_model：这张表，在xml进行部署时，它没有内容(flowable应该是放弃了此表改用act\_de\_model）。

四、流程模型的常见管理接口

1.部署api

2.验证部署是否成功/可以用来验证部署是否存在 api

3.获取所有部署api

4.删除部署api

5.部署的挂起与激活（以后还会讲到流程实例的挂起与激活，注意区分）

# flowable流程实例管理接口

一、流程实例的管理接口常见功能：

1.启动一个流程（即创建一个流程实例）

2.本人发起的流程实例

3.本人参与的流程实例

4.挂起/激活流程实例

5.提前终止流程实例（或又叫撤销流程）

6.彻底删除流程实例

7.流程图查看

8.判断流程实例的状态(运行中/已结束）

二、主要接口：

1.启动流程：

public ProcessInstance strartFlow(String processKey, Map<String, Object> paras){

//启动前必须开启,这样才能取得流程发起人

String userId=paras.get("AuthenticatedUserId").toString();

//设置流程实例的发起人是当前用户

Authentication.setAuthenticatedUserId(userId);

// 取得业务id

String busiKey = paras.get("busiKey").toString();

if (StringUtils.isEmpty(processKey)){

logger.error("错误：流程定义未找到！");

}

if (null == paras){

paras = new HashMap<>();

}

Deployment deployment = repositoryService.createDeploymentQuery().processDefinitionKey(processKey).singleResult();

if (deployment == null) {

logger.error("错误：启动流程实例:" + processKey);

}

ProcessInstance processInstance = runtimeService.startProcessInstanceByKey(processKey, busiKey, paras); // 同时设置业务的id到流程实例中的BUSINESS\_KEY\_

logger.info("成功：启动流程实例：" + processInstance.getProcessDefinitionName() + ":" + processInstance.getId());

Authentication.setAuthenticatedUserId(null); // 这个方法最终使用一个ThreadLocal类型的变量进行存储，也就是与当前的线程绑定，所以流程实例启动完毕之后，需要设置为null，防止多线程的时候出问题。

return processInstance;

}

2.提前终止流程实例：

public void deleteProcessInstanceById(String processInstanceId){

// ""这个参数本来可以写删除原因

runtimeService.deleteProcessInstance(processInstanceId,"");

}

3.本人发起的流程实例：

/\*\*

\* @Description: 我发起的流程实例列表

\*

\*

\* @Date: 2019/5/12 20:31

\* @Param:userId 用户id

\* @return: List<Zjm\_gflowTask> 流程实例列表

\*\*/

public List<HistoricProcessInstance> getMyStartProcint(String userId){

List<HistoricProcessInstance> list=processEngine.getHistoryService() // 历史相关Service

.createHistoricProcessInstanceQuery()

// .finished() 已完成的 unfinish 未完成的，或者不加表示全部

.startedBy(userId)

.orderByProcessInstanceStartTime().asc()

.list();

return list;

}

3.1 本人参与的流程实例：

//查询指定用户参与的流程信息 （流程历史  用户参与 ）

List hpis = historyService

.createHistoricProcessInstanceQuery().involvedUser(name)

.orderByProcessInstanceStartTime().desc().listPage(firstResult, maxResults);

4.挂起与激活流程实例：

/\*\*

\* @Description: 挂起流程实例

\*

\*

\* @Date: 2019/5/15 1:19

\* @Param: processInstanceId 当前流程实例id

\* @return: 无

\*\*/

public void handUpProcessInstance(String processInstanceId){

runtimeService.suspendProcessInstanceById(processInstanceId);

}

/\*\*

\* @Description:恢复（唤醒）被挂起的流程实例

\*

\*

\* @Date: 2019/5/15 1:22

\* @Param: processInstanceId 流程实例id

\* @return: 无

\*\*/

public void activateProcessInstance(String processInstanceId){

runtimeService.activateProcessInstanceById(processInstanceId);

}

5.生成流程图

// 将生成图片放到文件夹下

String deploymentId = "801";

// 获取图片资源的名称

List<String > list = processEngine.getRepositoryService().getDeploymentResourceNames(deploymentId);

// 定义图片资源的名称

String resourceName = "";

if (list !=null && list.size()>0){

for (String name:list){

if (name.indexOf(".png")>=0){

resourceName = name;

}

}

}

// 获取图片的输入流

InputStream in = processEngine.getRepositoryService().getResourceAsStream(deploymentId,resourceName);

// 将图片生成到D盘的根目录下

File file = new File ("D:/" + resourceName);

// 将输入流的图片写到d盘下

FileUtils.copyInputStreamToFile(in,file);

5.判断流程实例在运行中是否存在

/\*\*

\* @Description: 判断流程实例在运行中存不存在

\*

\*

\* @Date: 2019/5/26 19:24

\* @Param: procInstanceId 流程实例

\* @return: true表示存在,false表示不存在

\*\*/

public Boolean isExistProcIntRunning(String procInstanceId){

ProcessInstance pi= runtimeService.createProcessInstanceQuery().processInstanceId(procInstanceId).singleResult();

if (pi==null){

return false;

}else{

return true;

}

}

6.查询历史实例中流程实例是否存在

/\*\*

\* @Description: 查询历史实例中流程实例是否存在

\*

\*

\* @Date: 2019/5/26 19:35

\* @Param: procInstanceId 流程实例

\* @return: true表示存在,false表示不存在

\*\*/

public Boolean isExistProcHistory(String procInstanceId){

HistoricProcessInstance hpi= historyService.createHistoricProcessInstanceQuery().processInstanceId(procInstanceId).singleResult();

if (hpi==null){

return false;

}else{

return true;

}

}

# flowable任务实例管理接口

一、任务的管理功能主要有：

1.任务办理模式分两种（一家公司最好约定好需要不要签收模式，统一操作模式），一种直接办理模式，一种需要签收后再办理,所以分为：

    候选人的任务签收/候选组的任务签收/完成任务

2.转办任务

3.委派任务与代办任务

4.待办任务列表（不要签收的或者已签收未办理的）

5待办签收列表

6.抄送任务

7.终止任务（这个功能放到流程实例中）

8.催办任务

9.退回上一步（即驳回），退回到（历史某一个节点）

10.超时自动跳转完成

11.任务按业务类型划分：用户任务/服务任务/定时任务/会签任务/并行网关任务/排他网关任务

12.获取下一个节点集合（一个或者多个），获取节点名称，节点配置的用户/候选人员/候选组。

二、主要接口有：

1.候选人待签收任务列表：List<Task> tasks = taskService.createTaskQuery().taskCandidateUser(userId).list();

2.候选组待签收任务列表： List<Task> tasks = taskService.createTaskQuery().taskCandidateGroup(userGroup).list();

3.签收动作：taskService.claim(task.getId(), userId);//完成签收这个时候就任务Task的Assignee属性就有了值

4.已经签收过的任务列表，某种意义上才有真正的办理人List<Task> tasks = taskService.createTaskQuery().taskAssignee(userId).list();

5.完成任务

未签收/待办理 中取得需要办理的任务Task之后

 //调用 task.getId() = taskId 完成任务的办理，taskService服务自动流转到下一个办理节点

 taskService.complete(task.getId());

 //taskService.complete()方法 提供了重载，提供了3种方式供你选择

 complete(String taskId)

 complete(String taskId, Map<String,Object> variables)

 complete(String taskId, Map<String,Object> variables, boolean localScope)

6.任务并不一定要先签收再办理，可以直接到待办理中，而不是待签收中

 可以把

 taskService.claim(task.getId(), userId);方法

 替换为

 setAssignee(String taskId, String userId)

 经过以上步骤，任务已经在 办理中状态了

10.转办任务

taskService.setOwner(taskId, curUserId);

taskService.setAssignee(taskId,acceptUserId );

11.委派任务：是将任务节点分给其他人处理，等其他人处理好之后，委派任务会自动回到委派人的任务中

taskService.setOwner(taskId, curUserId);

taskService.delegateTask(taskId,acceptUserId);

12.被委派任务的办理:   办理完成后，委派任务会自动回到委派人的任务中

taskService.resolveTask(taskId, variables);

# flowable驳回/退回上一步/退回到

一、驳回/退回上一步/退回到（历史某一个节点）

        我们经常需要工作流中退回上一步，或者退回历史某一个节点。但由于流程的场景是很复杂的，回退有以下一些场景：

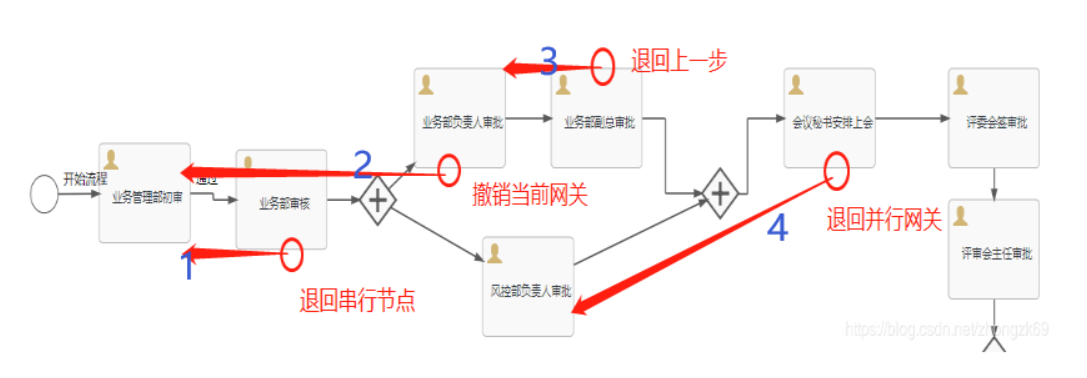
1.串行路线上的退回：流程中没有任何网关（排他网关/并行网关）和会签多实例。

2.退回到并行网关分支中的某一个节点上：

3.并行网关中的某一个分支节点上发起退回，退回到并行网关前面的某一个节点上

4.子流程中退回到主干流程中某一个节点/主干流程退回到子流程中某一个节点。

如下图：



二、flowable实现：

1.普通串行路线上的退回（此流程中没有并行网关的退回时），此方法支持普通串行节点/会签多实例节点/排他网关节点：

runtimeService.createChangeActivityStateBuilder()

.processInstanceId(proInstanceId)

.moveActivityIdsToSingleActivityId(curTaskKeys, targetTaskKey)

.changeState();

或者

moveActivityIdTo(String currentActivityId,String newActivityId)；

2.并行网关中发起退回（即撤销当前的网关）,这个地方不能用moveActivityIdTo(String currentActivityId,String newActivityId)；是因为当某一个分支完成，它的is\_active为0，另一条分支没有完成时。这时候这个方法是取不到所的分支的key的，它只有is\_active为1的key能取到，不然就会造成多一条垃圾数据，同时再走并行时，任何一个分支不会等另一个分支就完走到分支的合并节点上，这就是bug，所以要改为以下方法：

// 并行网关的退回

List<String > currentExecutionIds = new ArrayList<>();

List<Execution> executions = runtimeService.createExecutionQuery().parentId(proInstanceId).list();

for (Execution execution : executions) {

System.out.println("并行网关节点数："+execution.getActivityId());

currentExecutionIds.add(execution.getId());

}

runtimeService.createChangeActivityStateBuilder()

.moveExecutionsToSingleActivityId(currentExecutionIds, targetTaskKey)

.changeState();

3.退回到并行网关中的某一个节点：经试验退回时必须同时退回并行网关中的所有分支。

List<String > targetTaskKeys = new ArrayList<>();

targetTaskKeys.add("sid-CA74ADED-7E70-451D-951C-95988BFC3F07");

targetTaskKeys.add("sid-7922C598-74FD-4848-95AC-D9790AF68432");

runtimeService.createChangeActivityStateBuilder()

.processInstanceId(proInstanceId)

.moveSingleActivityIdToActivityIds("sid-CAD50E6F-7E0C-437D-816B-DDBA1A976A79", targetTaskKeys)

.changeState();

4.主干流程和子流程的退回（没有试验过），官方提供了以下方法：

moveActivityIdToParentActivityId(String currentActivityId, String newActivityId)

moveActivityIdToSubProcessInstanceActivityId(String currentActivityId, String newActivityId, String callActivityId)

​

moveActivityIdToSubProcessInstanceActivityId(String currentActivityId,  String newActivityId, String callActivityId,Integer subProcessDefinitionVersion)

# flowable的意见表和附件表应用

一、act\_hi\_comment和act\_hi\_attachment两表表，前者意见表后者是附件表

意见表它里面有一个类型type字段，分别是comment代表意见(这个type可以自定义,例如自定义为：通过/退回/提前终止），缺省提供了comment，event是事件，当你给附件表增加记录时，它就会在意见表同时增加多少条记录，并且type的类型为event。所以取意见时，需要注意这个问题。

（一）act\_hi\_comment提供了以下方法：

/\*\* Add a comment to a task and/or process instance. \*/

Comment addComment(String taskId, String processInstanceId, String message);

/\*\* Add a comment to a task and/or process instance with a custom type. \*/

Comment addComment(String taskId, String processInstanceId, String type, String message);

/\*\* Update a comment to a task and/or process instance. \*/

void saveComment(Comment comment);

/\*\*

\* Returns an individual comment with the given id. Returns null if no comment exists with the given id.

\*/

Comment getComment(String commentId);

/\*\* Removes all comments from the provided task and/or process instance \*/

void deleteComments(String taskId, String processInstanceId);

/\*\*

\* Removes an individual comment with the given id.

\*

\* @throws FlowableObjectNotFoundException

\* when no comment exists with the given id.

\*/

void deleteComment(String commentId);

/\*\* The comments related to the given task. \*/

List<Comment> getTaskComments(String taskId);

/\*\* The comments related to the given task of the given type. \*/

List<Comment> getTaskComments(String taskId, String type);

/\*\* All comments of a given type. \*/

List<Comment> getCommentsByType(String type);

/\*\* The all events related to the given task. \*/

List<Event> getTaskEvents(String taskId);

/\*\*

\* Returns an individual event with the given id. Returns null if no event exists with the given id.

\*/

Event getEvent(String eventId);

/\*\* The comments related to the given process instance. \*/

List<Comment> getProcessInstanceComments(String processInstanceId);

/\*\* The comments related to the given process instance. \*/

List<Comment> getProcessInstanceComments(String processInstanceId, String type);

（二）附件提供了以下方法：

/\*\*

\* Add a new attachment to a task and/or a process instance and use an input stream to provide the content

\*/

Attachment createAttachment(String attachmentType, String taskId, String processInstanceId, String attachmentName, String attachmentDescription, InputStream content);

/\*\*

\* Add a new attachment to a task and/or a process instance and use an url as the content

\*/

Attachment createAttachment(String attachmentType, String taskId, String processInstanceId, String attachmentName, String attachmentDescription, String url);

/\*\* Update the name and description of an attachment \*/

void saveAttachment(Attachment attachment);

/\*\* Retrieve a particular attachment \*/

Attachment getAttachment(String attachmentId);

/\*\* Retrieve stream content of a particular attachment \*/

InputStream getAttachmentContent(String attachmentId);

/\*\* The list of attachments associated to a task \*/

List<Attachment> getTaskAttachments(String taskId);

/\*\* The list of attachments associated to a process instance \*/

List<Attachment> getProcessInstanceAttachments(String processInstanceId);

/\*\* Delete an attachment \*/

void deleteAttachment(String attachmentId);

(三）真实应用：





# flowable核心数据库表清单

一.工作流部署—RepositoryService

1.流程定义数据表：act\_re\_procdef

2.流程设计模型部署：act\_re\_model

3.部署信息表：act\_re\_deployment

二.工作流运行表—RuntimeService

1.运行时流程人员表：act\_ru\_identitylink

2.Event时间监听信息表：act\_ru\_event\_subscr

3.运行时流程执行实例表：act\_ru\_execution

4.运行时定时任务数据表：act\_ru\_job

5.运行时任务节点表：act\_ru\_task

6.运行时流程变量数据表：act\_ru\_variable

三.工作流历史表 — HistoryService

1.历史节点表：act\_hi\_actinst 记录流程流转过的所有节点

2.历史任务实例表：act\_hi\_taskinst 只记录usertask内容

3.历史附件表：act\_hi\_attachment

4.历史意见表：act\_hi\_comment

5.历史详情表：act\_hi\_detail 流程中产生的变量详情

5.历史变量表：act\_hi\_varinst

四.工作流组织机构表 — IdentityService

1.用户信息表：act\_id\_user

2.用户扩展信息表：act\_id\_info

3.用户组信息表：act\_id\_group

4.用户与用户组对应信息表：act\_id\_membership

五.全局属性表

1.二进制数据表：act\_ge\_bytearray

2.全局属性：act\_ge\_property

# flowable核心数据库表详细表字段说明

数据模型设计

清单

| 数据表分类 | 描述 |
| --- | --- |
| ACT\_GE\_\* | 通用数据表 |
| ACT\_RE\_\* | 流程定义存储表 |
| ACT\_ID\_\* | 身份信息表 |
| ACT\_RU\_\* | 运行时数据库表 |
| ACT\_HI\_\* | 历史数据库表 |

通用数据库

| 数据表 | 描述 |
| --- | --- |
| ACT\_GE\_PROPERTY | 属性表(保存流程引擎的kv键值属性)--PropertyEntityImpl |
| ACT\_GE\_BTYEARRAY | 资源表(存储流程定义相关的资源)--ByteArrayEntityImpl |

流程定义存储表

| 数据表 | 描述 |
| --- | --- |
| ACT\_RE\_DEPLOYMENT | 流程部署表--DeploymentEntityImpl |
| ACT\_RE\_PROCDEF | 流程定义信息表--ProcessDefinitionEntityImpl |
| ACT\_RE\_MODEL | 模型信息表(用于Web设计器)--ModelEntityImpl |
| ACT\_PROCDEF\_INFO | 流程定义动态改变信息表--ProcessDefinitionInfoEntityImpl |

身份数据表

| 数据表 | 描述 |
| --- | --- |
| ACT\_ID\_USER | 用户基本信息表--UserEntityImpl |
| ACT\_ID\_INFO | 用户扩展表--IdentityInfoEntityImpl |
| ACT\_ID\_GROUP | 群组表(用于Web设计器)--GroupEntityImpl |
| ACT\_ID\_MEMBERSHIP | 用户与群主关系表--MemberShipEntityImpl |
| ACT\_ID\_BYTEARRAY | 二进制数据表(flowable)-- |
| ACT\_ID\_PRIV | 权限表(flowable)-- |
| ACT\_ID\_PRIV\_MAPPING | 用户或组权限关系表(flowable)-- |
| ACT\_ID\_PROPERTY | 属性表(flowable)-- |
| ACT\_ID\_TOKEN | 系统登录日志表(flowable)-- |

运行时流程数据表

| 数据表 | 描述 |
| --- | --- |
| ACT\_RU\_EXECUTION | 流程实例与分支执行表--ExecutionEntityImpl |
| ACT\_RU\_TASK | 用户任务表--TaskEntityImpl |
| ACT\_RU\_VARIABLE | 变量信息--VariableInstanceEntityImpl |
| ACT\_RU\_IDENTITYLINK | 参与者相关信息表--IdentityLinkEntityImpl |
| ACT\_RU\_EVENT\_SUBSCR | 事件订阅表--EventSubscriptionEntityImpl |
| ACT\_RU\_JOB | 作业表--JobEntityImpl |
| ACT\_RU\_TIMER\_JOB | 定时器表--TimerJobEntityImpl |
| ACT\_RU\_SUSPENDED\_JOB | 暂停作业表--SuspendedJobEntityImpl |
| ACT\_RU\_DEADLETTER\_JOB | 死信表--DeadLetterJobEntityImpl |
| ACT\_RU\_HISTORY\_JOB | 历史作业表(flowable)-- |

历史流程数据表

| 数据表 | 描述 |
| --- | --- |
| ACT\_HI\_PROCINST | 历史流程实例表--HistoricProcessInstanceEntityImpl |
| ACT\_HI\_ACTINST | 历史节点信息表--HistoricActivityInstanceEntityImpl |
| ACT\_HI\_TASKINST | 历史任务表--HistoricTaskInstanceEntityImpl |
| ACT\_HI\_VARINST | 历史变量--HistoricVariableInstanceEntityImpl |
| ACT\_HI\_IDENTITYLINK | 历史参与者表--HistoricIdentityLinkEntityImpl |
| ACT\_HI\_DETAIL | 历史的流程运行中的细节信息--HistoricDetailEntityImpl |
| ACT\_HI\_ATTACHMENT | 附件表--AttachmentEntityImpl |
| ACT\_HI\_COMMENT | 评论表--CommentEntityImpl |
| ACT\_EVT\_LOG | 事件日志表--EventLogEntryEntityImpl |

通用类

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_ge\_property（全局配置文件） | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| NAME\_ | 名称 |  | NO | varchar | 64 | PRI | schema.version schema.history next.dbid |
| VALUE\_ | 值 | NULL | YES | varchar | 300 |  | 5.\* create(5.\*) |
| REV\_ | 版本号 | NULL | YES | int | NULL |  | version |
| 注： 1.全局参数， 默认三个参数next.dbid， IdGenerator区间， schema.history， 自动执行sql历史， schema.version， 当 前sql版本。 2.属性数据表。存储整个流程引擎级别的数据。 | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_ge\_bytearray（二进制文件） | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| ID\_ | 主键 |  | NO | varchar | 64 | PRI |  |
| REV\_ | 版本号 | NULL | YES | int | NULL |  | version |
| NAME\_ | 名称 | NULL | YES | varchar | 255 |  | 部署的文件名称，如：mail.bpmn、mail.png 、mail.bpmn20.xml |
| DEPLOYMENT\_ID\_ | 部署ID | NULL | YES | varchar | 64 | ACT\_RE\_DEPLOYMENT |  |
| BYTES\_ | 字节（二进制数据） | NULL | YES | longblob | 4294967295 |  |  |
| GENERATED\_ | 是否系统生成 | NULL | YES | tinyint | NULL |  | 0为用户上传， 1为系统自动生 成， 比如系统会 自动根据xml生 成png |
| 注： 1.用来保存部署文件的大文本数据 2.所有二进制内容都会保存在这个表里， 比如部署的process.bpmn20.xml, process.png, user.form, 附件， bean序列 化为二进制的流程变量。  <p>3. act\_ge\_property属性数据表存储整个流程引擎级别的数据,初始化表结构时，会默认插入三条记录。</p>          </td>      </tr></tbody></table> | | | | | | | |

历史类

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_hi\_actinst（历史节点表） | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| ID\_ | 主键 | NULL | NO | varchar | 64 | PRI |  |
| PROC\_DEF\_ID\_ | 流程定义ID | NULL | NO | varchar | 64 |  |  |
| PROC\_INST\_ID\_ | 流程实例ID | NULL | NO | varchar | 64 | MUL |  |
| ACT\_ID\_ | 节点ID | NULL | NO | varchar | 255 |  |  |
| TASK\_ID\_ | 任务ID | NULL | YES | varchar | 64 |  | 任务实例ID 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 | NULL | YES | varchar | 64 |  |  |
| ACT\_NAME\_ | 节点名称 | NULL | YES | varchar | 255 |  |  |
| ACT\_TYPE\_ | 节点类型 | NULL | NO | varchar | 255 |  | 如startEvent、userTask |
| ASSIGNEE\_ | 签收人 | NULL | YES | varchar | 255 |  | 经办人 |
| START\_TIME\_ | 开始时间 | NULL | NO | datetime | NULL | MUL |  |
| END\_TIME\_ | 结束时间 | NULL | YES | datetime | NULL | MUL |  |
| DURATION\_ | 耗时 | NULL | YES | bigint | NULL |  | 毫秒值 |
| TENANT\_ID\_ | 多租户 |  | YES | varchar | 255 |  |  |
| 注： 1.　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。 2.  TENANT\_ID 是后续才加入的多租户 | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_hi\_attachment（历史附件表） | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| ID\_ | 主键ID | NULL | NO | varchar | 64 | PRI |  |
| REV\_ | 版本号 | NULL | YES | int | NULL |  | version |
| USER\_ID\_ | 用户ID | NULL | YES | varchar | 255 |  |  |
| NAME\_ | 名称 | NULL | YES | varchar | 255 |  |  |
| DESCRIPTION\_ | 描述 | NULL | YES | varchar | 4000 |  |  |
| TYPE\_ | 类型 | NULL | YES | varchar | 255 |  |  |
| TASK\_ID\_ | 任务ID | NULL | YES | varchar | 64 |  |  |
| PROC\_INST\_ID\_ | 流程实例ID | NULL | YES | varchar | 64 |  |  |
| URL\_ | 附件地址 | NULL | YES | varchar | 4000 |  | 附件的URL地址 |
| CONTENT\_ID\_ | 字节表ID | NULL | YES | varchar | 64 |  | ACT\_GE\_BYTEARRAY的ID |
| TIME\_ | 时间 | NULL | YES | datetime | NULL |  |  |
| 注： 1.存放历史流程相关的附件。 2.时间是后续版本加入 | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_hi\_comment（历史审批意见表） | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| ID\_ | 主键 | NULL | NO | varchar | 64 | PRI |  |
| TYPE\_ | 类型 | NULL | YES | varchar | 255 |  | 类型：event（事件） comment（意见） |
| TIME\_ | 时间 | NULL | NO | datetime | NULL |  |  |
| USER\_ID\_ | 用户ID | NULL | YES | varchar | 255 |  |  |
| TASK\_ID\_ | 任务ID | NULL | YES | varchar | 64 |  |  |
| PROC\_INST\_ID\_ | 流程实例ID | NULL | YES | varchar | 64 |  |  |
| ACTION\_ | 行为类型 | NULL | YES | varchar | 255 |  |  |
| MESSAGE\_ | 基本内容 | NULL | YES | varchar | 4000 |  | 用于存放流程产生的信息，比如审批意见 |
| FULL\_MSG\_ | 全部内容 | NULL | YES | longblob | 4294967295 |  | 附件 |
| 注： 1. 存放历史流程的审批意见。 2. 行为类型。值为下列内容中的一种：AddUserLink、DeleteUserLink、AddGroupLink、DeleteGroupLink、AddComment、AddAttachment、DeleteAttachment | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_hi\_detail（历史详情信息表） | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| ID\_ | 主键 | NULL | NO | varchar | 64 | PRI |  |
| TYPE\_ | 类型 | NULL | NO | varchar | 255 |  | 类型:    FormProperty,   //表单    VariableUpdate  //参数 |
| PROC\_INST\_ID\_ | 流程实例ID | NULL | YES | varchar | 64 | MUL |  |
| EXECUTION\_ID\_ | 执行实例 | NULL | YES | varchar | 64 |  |  |
| TASK\_ID\_ | 任务ID | NULL | YES | varchar | 64 | MUL |  |
| ACT\_INST\_ID\_ | 节点实例ID | NULL | YES | varchar | 64 | ACT\_HI\_ACTINST |  |
| NAME\_ | 名称 | NULL | NO | varchar | 255 | MUL |  |
| VAR\_TYPE\_ | 参数类型 | NULL | YES | varchar | 255 |  |  |
| REV\_ | 版本号 | NULL | YES | int | NULL |  | version |
| TIME\_ | 时间戳 | NULL | NO | datetime | NULL | MUL | 创建时间 |
| BYTEARRAY\_ID\_ | 字节表ID | NULL | YES | varchar | 64 | ACT\_GE\_BYTEARRAY |  |
| DOUBLE\_ | 浮点值 | NULL | YES | double | NULL |  | 存储变量类型为Double |
| LONG\_ | 长整型 | NULL | YES | bigint | NULL |  | 存储变量类型为long |
| TEXT\_ | 文本值 | NULL | YES | varchar | 4000 |  | 存储变量值类型为String |
| TEXT2\_ | 字符串 | NULL | YES | varchar | 4000 |  | 此处存储的是JPA持久化对象时，才会有值。此值为对象ID jpa变量text存className,text2存id |
| 注： 1.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。 2.参数类型：  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 | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_hi\_identitylink(历史流程人员表) | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| ID\_ | 主键 |  | NO | varchar | 64 | PRI |  |
| GROUP\_ID\_ | 用户组ID | NULL | YES | varchar | 255 |  |  |
| TYPE\_ | 类型 | NULL | YES | varchar | 255 |  | 类型，主要分为以下几种： assignee、candidate、owner、starter 、participant |
| USER\_ID\_ | 用户ID | NULL | YES | varchar | 255 | MUL |  |
| TASK\_ID\_ | 任务ID | NULL | YES | varchar | 64 | MUL |  |
| PROC\_INST\_ID\_ | 流程实例ID | NULL | YES | varchar | 64 | MUL |  |
| 注： 1.  任务参与者数据表。主要存储当前节点参与者的信息。 | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_hi\_procinst（流程实例历史\*核心表） | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| ID\_ | 主键 | NULL | NO | varchar | 64 | PRI |  |
| PROC\_INST\_ID\_ | 流程实例ID | NULL | NO | varchar | 64 | UNI |  |
| BUSINESS\_KEY\_ | 业务标识 | NULL | YES | varchar | 255 | MUL | 业务主键，业务表单的ID |
| PROC\_DEF\_ID\_ | 流程实例ID | NULL | NO | varchar | 64 |  |  |
| START\_TIME\_ | 开始时间 | NULL | NO | datetime | NULL |  |  |
| END\_TIME\_ | 结束时间 | NULL | YES | datetime | NULL | MUL |  |
| DURATION\_ | 耗时 | NULL | YES | bigint | NULL |  |  |
| START\_USER\_ID\_ | 流程发起人ID | NULL | YES | varchar | 255 |  |  |
| START\_ACT\_ID\_ | 开始节点ID | NULL | YES | varchar | 255 |  |  |
| END\_ACT\_ID\_ | 结束节点ID | NULL | YES | varchar | 255 |  |  |
| SUPER\_PROCESS\_INSTANCE\_ID\_ | 父流程实例ID | NULL | YES | varchar | 64 |  |  |
| DELETE\_REASON\_ | 删除原因 | NULL | YES | varchar | 4000 |  |  |
| TENANT\_ID\_ | 租户ID |  | YES | varchar | 255 |  |  |
| NAME\_ | 名称 | NULL | YES | varchar | 255 |  |  |
| 注： 1.核心表之一。 2.存放历史的流程实例。 3.设计历史流程实例表的初衷之一就是为了使得运行时库数据量尽可能小，效率最优。 | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_hi\_taskinst（历史任务流程实例信息\*核心表） | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| ID\_ | 主键 | NULL | NO | varchar | 64 | PRI |  |
| PROC\_DEF\_ID\_ | 流程实例ID | NULL | YES | varchar | 64 |  |  |
| TASK\_DEF\_KEY\_ | 任务节点定义ID | NULL | YES | varchar | 255 |  | 任务定义标识（环节ID） |
| PROC\_INST\_ID\_ | 流程实例ID | NULL | YES | varchar | 64 | MUL |  |
| EXECUTION\_ID\_ | 执行实例ID | NULL | YES | varchar | 64 |  |  |
| NAME\_ | 任务名称 | NULL | YES | varchar | 255 |  |  |
| PARENT\_TASK\_ID\_ | 父任务节点ID | NULL | YES | varchar | 64 |  |  |
| DESCRIPTION\_ | 描述 | NULL | YES | varchar | 4000 |  |  |
| OWNER\_ | 被代理人 | NULL | YES | varchar | 255 |  | 委托人（默认为空，只有在委托时才有值） |
| ASSIGNEE\_ | 经办人 | NULL | YES | varchar | 255 |  |  |
| START\_TIME\_ | 开始时间 | NULL | NO | datetime | NULL |  |  |
| CLAIM\_TIME\_ | 签收时间 | NULL | YES | datetime | NULL |  |  |
| END\_TIME\_ | 结束时间 | NULL | YES | datetime | NULL |  |  |
| DURATION\_ | 耗时 | NULL | YES | bigint | NULL |  |  |
| DELETE\_REASON\_ | 删除原因 | NULL | YES | varchar | 4000 |  | 删除原因(completed,deleted) |
| PRIORITY\_ | 优先级 | NULL | YES | int | NULL |  |  |
| DUE\_DATE\_ | 截止时间 | NULL | YES | datetime | NULL |  | 过期时间，表明任务应在多长时间内完成 |
| FORM\_KEY\_ | FORM表单的KEY | NULL | YES | varchar | 255 |  | desinger节点定义的 form\_key属性 |
| CATEGORY\_ | 分类 | NULL | YES | varchar | 255 |  |  |
| TENANT\_ID\_ | 租户ID |  | YES | varchar | 255 |  |  |
| 注： 1.　历史任务实例表。 2. 存放已经办理的任务。 3. CATEGORY和TNANT\_ID是后续版本才加进来的。 | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_hi\_varinst（历史变量表） | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| ID\_ | 主键 | NULL | NO | varchar | 64 | PRI |  |
| PROC\_INST\_ID\_ | 流程实例ID | NULL | YES | varchar | 64 | MUL |  |
| EXECUTION\_ID\_ | 执行实例ID | NULL | YES | varchar | 64 |  |  |
| TASK\_ID\_ | 任务ID | NULL | YES | varchar | 64 | MUL |  |
| NAME\_ | 名称 | NULL | NO | varchar | 255 | MUL |  |
| VAR\_TYPE\_ | 变量类型 | NULL | YES | varchar | 100 |  |  |
| REV\_ | 版本号 | NULL | YES | int | NULL |  | version |
| BYTEARRAY\_ID\_ | 字节流ID | NULL | YES | varchar | 64 | ACT\_GE\_BYTEARRAY |  |
| DOUBLE\_ | 浮点值 | NULL | YES | double | NULL |  | 存储DoubleType类型的数据 |
| LONG\_ | 长整型 | NULL | YES | bigint | NULL |  | 存储LongType类型的数据 |
| TEXT\_ | 文本值 | NULL | YES | varchar | 4000 |  | 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\_ | 文本值 | NULL | YES | varchar | 4000 |  |  |
| CREATE\_TIME\_ | 创建时间 | NULL | YES | datetime | NULL |  |  |
| LAST\_UPDATED\_TIME\_ | 最后更新时间 | NULL | YES | datetime | NULL |  |  |
| 注： 1. 主要存放历史变量数据。 | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_evt\_log（事件日志） | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| LOG\_NR\_ | 主键 | NULL | NO | bigint | NULL | PRI |  |
| TYPE\_ | 类型 | NULL | YES | varchar | 64 |  |  |
| PROC\_DEF\_ID\_ | 流程定义ID | NULL | YES | varchar | 64 |  |  |
| PROC\_INST\_ID\_ | 流程实例ID | NULL | YES | varchar | 64 |  |  |
| EXECUTION\_ID\_ | 执行ID | NULL | YES | varchar | 64 |  |  |
| TASK\_ID\_ | 任务ID | NULL | YES | varchar | 64 |  |  |
| TIME\_STAMP\_ | 时间 | CURRENT\_TIMESTAMP(3) | NO | timestamp | NULL |  |  |
| USER\_ID\_ | 用户ID | NULL | YES | varchar | 255 |  |  |
| DATA\_ | 数据 | NULL | YES | longblob | 4294967295 |  |  |
| LOCK\_OWNER\_ | 锁定节点 | NULL | YES | varchar | 255 |  |  |
| LOCK\_TIME\_ | 锁定时间 | NULL | YES | timestamp | NULL |  |  |
| IS\_PROCESSED\_ | 是否正在执行 | 0 | YES | tinyint | NULL |  |  |
| 注： 1.事件日志表 2.事件日志， 默认不开启。 3.从Activiti 5.16开始，引入了（试验性）的事件记录机制。记录机制基于Activiti引擎的事件机制的一般用途，并默认禁用。其思想是，来 源于引擎的事件会被捕获，并创建一个包含了所有事件数据（甚至更多）的映射，提供给 org.activiti.engine.impl.event.logger.EventFlusher，由它将这些数据刷入其他地方。默认情况下，使用简单的基于数据库的事件处理/刷 入，会使用Jackson将上述映射序列化为JSON，并将其作为EventLogEntryEntity接口存入数据库。如果不使用事件记录，可以删除这个 表。 4.配置启用事件日志：processEngineConfiguration.setEnableDatabaseEventLogging(true); 5.运行时启用事件日志：databaseEventLogger = new EventLogger(processEngineConfiguration.getClock());                       runtimeService.addEventListener(databaseEventLogger); 6.可以扩展EventLogger类。如果默认的数据库记录不符合要求，需要覆盖createEventFlusher()方法返回一 个org.activiti.engine.impl.event.logger.EventFlusher接口的实例。可以通过Activiti的 managementService.getEventLogEntries(startLogNr, size)?获取EventLogEntryEntity实例。 容易看出这个表中的数据可以通过JSON放入大数据NoSQL存储，例如MongoDB，Elastic Search，等等。也容易看出这里使用的类 （org.activiti.engine.impl.event.logger.EventLogger/EventFlusher与许多其他 EventHandler类）是可插入的，可以按你的使用场景调整 （例如不将JSON存入数据库，而是将其直接发送给一个队列或大数据存储）。 请注意这个事件记录机制是额外于Activiti的“传统”历史管理器的。尽管所有数据都在数据库表中，但并未对查询或快速恢复做优化。实 际使用场景是末端审计并将其存入大数据存储。 | | | | | | | |

用户身份类

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_id\_group（用户组） | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| ID\_ | 主键 |  | NO | varchar | 64 | PRI |  |
| REV\_ | 版本号 | NULL | YES | int | NULL |  | version |
| NAME\_ | 名称 | NULL | YES | varchar | 255 |  |  |
| TYPE\_ | 类型 | NULL | YES | varchar | 255 |  |  |
| 注： 1.Activiti自带的用户组表，用于组任务。 | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_id\_info（用户扩展信息表） | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| ID\_ | 主键 |  | NO | varchar | 64 | PRI |  |
| REV\_ | 版本号 | NULL | YES | int | NULL |  | version |
| USER\_ID\_ | 用户ID | NULL | YES | varchar | 64 |  |  |
| TYPE\_ | 类型 | NULL | YES | varchar | 64 |  |  |
| KEY\_ | 属性名 | NULL | YES | varchar | 255 |  |  |
| VALUE\_ | 属性值 | NULL | YES | varchar | 255 |  |  |
| PASSWORD\_ | 密码 | NULL | YES | longblob | 4294967295 |  |  |
| PARENT\_ID\_ | 父级ID | NULL | YES | varchar | 255 |  |  |
| 注： | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_id\_membership（ 用户与分组对应信息表） | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| USER\_ID\_ | 用户ID |  | NO | varchar | 64 | PRI（ACT\_ID\_USER） |  |
| GROUP\_ID\_ | 用户组ID |  | NO | varchar | 64 | PRI（ACT\_ID\_GROUP） |  |
| 注： 1.用来保存用户的分组信息。 | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_id\_user（用户信息表） | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| ID\_ | 主键 |  | NO | varchar | 64 | PRI |  |
| REV\_ | 版本号 | NULL | YES | int | NULL |  | version |
| FIRST\_ | 姓 | NULL | YES | varchar | 255 |  | FIRST\_NAME |
| LAST\_ | 名 | NULL | YES | varchar | 255 |  | LAST\_NAME |
| EMAIL\_ | 邮箱 | NULL | YES | varchar | 255 |  |  |
| PWD\_ | 密码 | NULL | YES | varchar | 255 |  |  |
| PICTURE\_ID\_ | 头像ID | NULL | YES | varchar | 64 | ACT\_GE\_BYTEARRAY |  |
| 注： 1.Activiti用户信息表。 | | | | | | | |

流程定义存储表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_procdef\_info（流程定义更新信息） | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| ID\_ | 主键 | NULL | NO | varchar | 64 | PRI |  |
| PROC\_DEF\_ID\_ | 流程定义ID | NULL | NO | varchar | 64 | UNI（ACT\_RE\_PROCDEF） |  |
| REV\_ | 版本号 | NULL | YES | int | NULL |  | version |
| INFO\_JSON\_ID\_ | 内容 | NULL | YES | varchar | 64 | MUL（ACT\_GE\_BYTEARRAY） |  |
| 注： 1.流程版本升级的数据。 | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_re\_deployment（  部署信息表\*核心表） | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| ID\_ | 主键 |  | NO | varchar | 64 | PRI |  |
| NAME\_ | 名称 | NULL | YES | varchar | 255 |  |  |
| CATEGORY\_ | 分类 | NULL | YES | varchar | 255 |  |  |
| TENANT\_ID\_ | 租户ID |  | YES | varchar | 255 |  |  |
| DEPLOY\_TIME\_ | 部署时间 | NULL | YES | timestamp | NULL |  |  |
| 注： 1.　部署流程定义时需要被持久化保存下来的信息。 | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_re\_model( 流程设计模型部署表) | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| ID\_ | 主键 | NULL | NO | varchar | 64 | PRI |  |
| REV\_ | 版本号 | NULL | YES | int | NULL |  | version |
| NAME\_ | 名称 | NULL | YES | varchar | 255 |  |  |
| KEY\_ | 标识 | NULL | YES | varchar | 255 |  |  |
| CATEGORY\_ | 分类 | NULL | YES | varchar | 255 |  |  |
| CREATE\_TIME\_ | 创建时间 | NULL | YES | timestamp | NULL |  |  |
| LAST\_UPDATE\_TIME\_ | 最后更新时间 | NULL | YES | timestamp | NULL |  |  |
| VERSION\_ | 版本 | NULL | YES | int | NULL |  |  |
| META\_INFO\_ | 元数据 | NULL | YES | varchar | 4000 |  | 以json格式保存流程定义的信息 |
| DEPLOYMENT\_ID\_ | 部署ID | NULL | YES | varchar | 64 | MUL(ACT\_RE\_DEPLOYMENT） |  |
| EDITOR\_SOURCE\_VALUE\_ID\_ | 二进制文件ID | NULL | YES | varchar | 64 | MUL（ACT\_GE\_BYTEARRAY） | 设计器原始信息 |
| EDITOR\_SOURCE\_EXTRA\_VALUE\_ID\_ | 二进制文件ID | NULL | YES | varchar | 64 | MUL（ACT\_GE\_BYTEARRAY） | 设计器扩展信息 |
| TENANT\_ID\_ | 租户ID |  | YES | varchar | 255 |  |  |
| 注： 1.该表是流程设计器设计流程模型保存的数据。 | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_re\_procdef（流程定义数据表\*核心表） | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| ID\_ | 主键 | NULL | NO | varchar | 64 | PRI |  |
| REV\_ | 版本号 | NULL | YES | int | NULL |  | version |
| CATEGORY\_ | 分类 | NULL | YES | varchar | 255 |  | 流程定义的Namespace就是类别 |
| NAME\_ | 名称 | NULL | YES | varchar | 255 |  |  |
| KEY\_ | 标识 | NULL | NO | varchar | 255 | MUL |  |
| VERSION\_ | 版本 | NULL | NO | int | NULL |  |  |
| DEPLOYMENT\_ID\_ | 部署ID | NULL | YES | varchar | 64 |  |  |
| RESOURCE\_NAME\_ | 资源名称 | NULL | YES | varchar | 4000 |  | 流程bpmn文件名称 |
| DGRM\_RESOURCE\_NAME\_ | 图片资源名称 | NULL | YES | varchar | 4000 |  |  |
| DESCRIPTION\_ | 描述 | NULL | YES | varchar | 4000 |  |  |
| HAS\_START\_FORM\_KEY\_ | 拥有开始表单标识 | NULL | YES | tinyint | NULL |  | start节点是否存在formKey 0否  1是 |
| HAS\_GRAPHICAL\_NOTATION\_ | 拥有图形信息 | NULL | YES | tinyint | NULL |  |  |
| SUSPENSION\_STATE\_ | 挂起状态 | NULL | YES | int | NULL |  | 暂停状态 1激活 2暂停 |
| TENANT\_ID\_ | 租户ID |  | YES | varchar | 255 |  |  |
| 注： 1.　业务流程定义数据表。此表和ACT\_RE\_DEPLOYMENT是多对一的关系，即，一个部署的bar包里可能包含多个流程定义文件，每个流程定义文件都会有一条记录在ACT\_REPROCDEF表内，每个流程定义的数据，都会对于ACT\_GE\_BYTEARRAY表内的一个资源文件和PNG图片文件。和ACT\_GE\_BYTEARRAY的关联是通过程序用ACT\_GE\_BYTEARRAY.NAME与ACT\_RE\_PROCDEF.NAME\_完成的，在数据库表结构中没有体现。 | | | | | | | |

运行时流程数据表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_ru\_event\_subscr（事件订阅） | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| ID\_ | 主键 | NULL | NO | varchar | 64 | PRI |  |
| REV\_ | 版本号 | NULL | YES | int | NULL |  | varsion |
| EVENT\_TYPE\_ | 事件类型 | NULL | NO | varchar | 255 |  |  |
| EVENT\_NAME\_ | 事件名称 | NULL | YES | varchar | 255 |  |  |
| EXECUTION\_ID\_ | 执行实例ID | NULL | YES | varchar | 64 | MUL（ACT\_RU\_EXECUTION） |  |
| PROC\_INST\_ID\_ | 流程实例ID | NULL | YES | varchar | 64 |  |  |
| ACTIVITY\_ID\_ | 节点ID | NULL | YES | varchar | 64 |  |  |
| CONFIGURATION\_ | 配置 | NULL | YES | varchar | 255 | MUL |  |
| CREATED\_ | 创建时间 | CURRENT\_TIMESTAMP(3) | NO | timestamp | NULL |  |  |
| PROC\_DEF\_ID\_ | 流程定义ID | NULL | YES | varchar | 64 |  |  |
| TENANT\_ID\_ | 租户ID |  | YES | varchar | 255 |  |  |
| 注： 1.该表是后续版本加进来的。 | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_ru\_execution（运行时流程执行实例表\*核心表） | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| ID\_ | 主键 |  | NO | varchar | 64 | PRI |  |
| REV\_ | 版本号 | NULL | YES | int | NULL |  |  |
| PROC\_INST\_ID\_ | 流程实例ID | NULL | YES | varchar | 64 | MUL（ACT\_RU\_EXECUTION） |  |
| BUSINESS\_KEY\_ | 业务标识 | NULL | YES | varchar | 255 | MUL |  |
| PARENT\_ID\_ | 父级ID | NULL | YES | varchar | 64 | MUL（ACT\_RU\_EXECUTION） |  |
| PROC\_DEF\_ID\_ | 流程定义ID | NULL | YES | varchar | 64 | MUL（ACT\_RE\_PROCDEF） |  |
| SUPER\_EXEC\_ | 父流程实例中对应的 执行 | NULL | YES | varchar | 64 | MUL（ACT\_RU\_EXECUTION） |  |
| ACT\_ID\_ | 节点ID | NULL | YES | varchar | 255 |  |  |
| IS\_ACTIVE\_ | 是否激活 | NULL | YES | tinyint | NULL |  |  |
| IS\_CONCURRENT\_ | 是否分支（并行） | NULL | YES | tinyint | NULL |  | 是否为并行(true/false） |
| IS\_SCOPE\_ | 是否处于多实例或环 节嵌套状态 | NULL | YES | tinyint | NULL |  |  |
| IS\_EVENT\_SCOPE\_ | 是否激活状态 | NULL | YES | tinyint | NULL |  |  |
| SUSPENSION\_STATE\_ | 挂起状态 | NULL | YES | int | NULL |  | 暂停状态 1激活 2暂停 |
| CACHED\_ENT\_STATE\_ | 缓存状态 | NULL | YES | int | NULL |  | 缓存的状态， 1 事件 监听 2 人工任务 3 异 步作业 |
| TENANT\_ID\_ | 租户ID |  | YES | varchar | 255 |  |  |
| NAME\_ | 名称 | NULL | YES | varchar | 255 |  |  |
| LOCK\_TIME\_ | 锁定时间 | NULL | YES | timestamp | NULL |  |  |
| 注： 1.TENANT\_ID、NAME、LOCK\_TIME是后续版本加入的。 | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_ru\_identitylink（ 运行时流程人员表） | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| ID\_ | 主键 |  | NO | varchar | 64 | PRI |  |
| REV\_ | 版本号 | NULL | YES | int | NULL |  | version |
| GROUP\_ID\_ | 用户组ID | NULL | YES | varchar | 255 | MUL |  |
| TYPE\_ | 类型 | NULL | YES | varchar | 255 |  |  |
| USER\_ID\_ | 用户ID | NULL | YES | varchar | 255 | MUL |  |
| TASK\_ID\_ | 任务ID | NULL | YES | varchar | 64 | MUL（ACT\_RU\_TASK） |  |
| PROC\_INST\_ID\_ | 流程实例ID | NULL | YES | varchar | 64 | MUL（ACT\_RU\_EXECUTION） |  |
| PROC\_DEF\_ID\_ | 流程定义ID | NULL | YES | varchar | 64 | MUL(ACT\_RE\_PROCDEF) |  |
| 注： 1.任务参与者数据表。主要存储当前节点参与者的信息。 | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_ru\_job(运行时定时任务数据表) | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| ID\_ | 主键 | NULL | NO | varchar | 64 | PRI |  |
| REV\_ | 版本号 | NULL | YES | int | NULL |  |  |
| TYPE\_ | 类型 | NULL | NO | varchar | 255 |  |  |
| LOCK\_EXP\_TIME\_ | 锁定过期时间 | NULL | YES | timestamp | NULL |  |  |
| LOCK\_OWNER\_ | 挂起者 | NULL | YES | varchar | 255 |  |  |
| EXCLUSIVE\_ | 是否唯一 | NULL | YES | tinyint | NULL |  |  |
| EXECUTION\_ID\_ | 执行实例ID | NULL | YES | varchar | 64 |  |  |
| PROCESS\_INSTANCE\_ID\_ | 流程实例ID | NULL | YES | varchar | 64 |  |  |
| PROC\_DEF\_ID\_ | 流程定义ID | NULL | YES | varchar | 64 |  |  |
| RETRIES\_ | 重试次数 | NULL | YES | int | NULL |  |  |
| EXCEPTION\_STACK\_ID\_ | 异常堆栈 | NULL | YES | varchar | 64 | MUL（ACT\_GE\_BYTEARRAY） |  |
| EXCEPTION\_MSG\_ | 异常信息 | NULL | YES | varchar | 4000 |  |  |
| DUEDATE\_ | 截止时间 | NULL | YES | timestamp | NULL |  |  |
| REPEAT\_ | 重复 | NULL | YES | varchar | 255 |  |  |
| HANDLER\_TYPE\_ | 处理器类型 | NULL | YES | varchar | 255 |  |  |
| HANDLER\_CFG\_ | 处理器配置 | NULL | YES | varchar | 4000 |  |  |
| TENANT\_ID\_ | 租户ID |  | YES | varchar | 255 |  |  |
| 注： 1.作业执行器数据。 2.需要启用JOB组件：JobExecutor 是管理一组线程的组件，这些线程用于触发定时器（包括后续的异步消息）。在单元测试场景下，使用多线程会很笨重。 因此API提供 ManagementService.createJobQuery 用于查询，以及 ManagementService.executeJob 用于执行作业。这样作业的执 行就可以在单元测试内部控制。为了避免作业执行器的干扰，可以将它关闭。 默认情况下， JobExecutor 在流程引擎启动时激活。当你不希望 JobExecutor 随流程引擎启动时，设置：  <property name="jobExecutorActivate" value="false" /> 3.11. 启用异步执行器 Async executor activation AsyncExecutor 是管理线程池的组件，这个线程池用于触发定时器与异步任务。 默认情况下，由于历史原因，当使用 JobExecutor 时， AsyncExecutor 不生效。然而我们建议使用新的 AsyncExecutor 代替 JobExecutor ，通过定义两个参数实现 <property name="asyncExecutorEnabled" value="true" /> <property name="asyncExecutorActivate" value="true" /> asyncExecutorEnabled参数用于启用异步执行器，代替老的作业执行器。 第二个参数asyncExecutorActivate命令Activiti引擎在启动时 启动异步执行器线程池。 | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_ru\_task（ 运行时任务节点表\*核心表） | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| ID\_ | 主键 |  | NO | varchar | 64 | PRI |  |
| REV\_ | 版本号 | NULL | YES | int | NULL |  | version |
| EXECUTION\_ID\_ | 执行实例ID | NULL | YES | varchar | 64 | MUL（ACT\_RU\_EXECUTION） |  |
| PROC\_INST\_ID\_ | 流程实例ID | NULL | YES | varchar | 64 | MUL（ACT\_RU\_EXECUTION） |  |
| PROC\_DEF\_ID\_ | 流程定义ID | NULL | YES | varchar | 64 | MUL（ACT\_RE\_PROCDEF） |  |
| NAME\_ | 名称 | NULL | YES | varchar | 255 |  |  |
| PARENT\_TASK\_ID\_ | 父任务ID | NULL | YES | varchar | 64 |  |  |
| DESCRIPTION\_ | 描述 | NULL | YES | varchar | 4000 |  |  |
| TASK\_DEF\_KEY\_ | 人物定义标识 | NULL | YES | varchar | 255 |  |  |
| OWNER\_ | 被代理人 | NULL | YES | varchar | 255 |  | （一般情况下为空，只有在委托时才有值） |
| ASSIGNEE\_ | 经办人 | NULL | YES | varchar | 255 |  | 签收人或者委托人 |
| DELEGATION\_ | 委托状态 | NULL | YES | varchar | 64 |  | 委托状态 PENDING 委托中， RESOLVED已处理 |
| PRIORITY\_ | 优先级 | NULL | YES | int | NULL |  |  |
| CREATE\_TIME\_ | 创建时间 | NULL | YES | timestamp | NULL | MUL |  |
| DUE\_DATE\_ | 截止时间 | NULL | YES | datetime | NULL |  |  |
| CATEGORY\_ | 分类 | NULL | YES | varchar | 255 |  |  |
| SUSPENSION\_STATE\_ | 挂起状态 | NULL | YES | int | NULL |  | 暂停状态 1激活 2暂停 |
| TENANT\_ID\_ | 租户ID |  | YES | varchar | 255 |  |  |
| FORM\_KEY\_ | 表单标识 | NULL | YES | varchar | 255 |  |  |
| 注： 1.运行时任务数据表 | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| act\_ru\_variable（ 运行时流程变量数据表\*核心表） | | | | | | | |
| 字段 | 字段名称 | 字段默认值 | 是否允许为空 | 数据类型 | 字段长度 | 键 | 备注 |
| ID\_ | 主键 | NULL | NO | varchar | 64 | PRI |  |
| REV\_ | 版本号 | NULL | YES | int | NULL |  | version |
| TYPE\_ | 类型 | NULL | NO | varchar | 255 |  | 见备注 |
| NAME\_ | 名称 | NULL | NO | varchar | 255 |  |  |
| EXECUTION\_ID\_ | 执行实例ID | NULL | YES | varchar | 64 | MUL（ACT\_RU\_EXECUTION） |  |
| PROC\_INST\_ID\_ | 流程实例ID | NULL | YES | varchar | 64 | MUL（ACT\_RU\_EXECUTION） |  |
| TASK\_ID\_ | 任务ID | NULL | YES | varchar | 64 | MUL（ACT\_RU\_TASK） |  |
| BYTEARRAY\_ID\_ | 资源ID | NULL | YES | varchar | 64 | MUL（ACT\_GE\_BYTEARRAY） |  |
| DOUBLE\_ | 浮点值 | NULL | YES | double | NULL |  | 存储变量类型为Double |
| LONG\_ | 长整型 | NULL | YES | bigint | NULL |  | 存储变量类型为long |
| TEXT\_ | 文本值 | NULL | YES | varchar | 4000 |  | 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\_ | 文本值 | NULL | YES | varchar | 4000 |  | 此处存储的是JPA持久化对象时，才会有值。此值为对象ID |
| 注： 1.运行时流程变量数据表。 2.类型：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 | | | | | | | |

# flowable数据库表中流程实例、活动实例、任务实例三者之间关系分析

场景模拟（请假流程）：

员工申请请假

部门领导审批

人事审批

员工销假

本文用次例介绍在工作流中出现的几个对象及其之间的关系，以及在Activiti中各个对象是如何关联的。

在线演示实例：<http://aws.kafeitu.me:8080/kft-activiti-demo>

在开始之前先看看下图，对整个对象结构有个了解，再结合下面的详细介绍理解。

图1 Activiti中几个对象之间的关系

**1.ProcessInstance**

员工开始申请请假流程，通过runtimeService.startProcessInstance()方法启动，引擎会创建一个**流程实例（ProcessInstance）**。

简单来说流程实例就是根据一次（一条）业务数据用**流程驱动的入口**，两者之间是**一对一**的关系。

擎会创建一条数据到**ACT\_RU\_EXECUTION**表，同时也会根据**history**的级别决定是否查询相同的历史数据到**ACT\_HI\_PROCINST**表。

启动完流程之后业务和流程已经建立了关联关系，第一步结束。

启动流程和业务关联区别：

对于**自定义表单**来说启动的时候会传入**businessKey**作为业务和流程的关联属性

对于**动态表单**来说不需要使用businessKey关联，因为所有的数据都保存在引擎的表中

对于**外部表单**来说businessKey是可选的，但是一般不会为空，和自定义表单类似

关于各种表单之间的区别请参考：[比较Activiti中三种不同的表单及其应用](http://www.kafeitu.me/activiti/2012/08/05/diff-activiti-workflow-forms.html)

**2.Execution**

初学者最搞不懂的就是ProcessInstance与Execution之间的关系，要分两种情况说明。

Execution的含义就是一个流程实例（ProcessInstance）具体要执行的过程对象。

不过在说明之前先声明两者的对象映射关系：

ProcessInstance（1）--->Execution(N)，其中N >= 1。

值相等的情况：

除了在流程中启动的子流程之外，流程启动之后在表ACT\_RU\_EXECUTION中的字段ID\_和PROC\_INST\_ID\_字段值是相同的。

**图2 ID\_和PROC\_INST\_ID\_相等**

值不相等的情况：

不相等的情况目前只会出现在子流程中（包含：嵌套、引入），例如一个购物流程中除了下单、出库节点之外可能还有一个付款子流程，在实际企业应用中付款流程通常是作为公用的，所以使用子流程作为主流程（购物流程）的一部分。

当任务到达子流程时引擎会自动创建一个付款流程，但是这个流程有一个特殊的地方，在数据库可以直观体现，如下图。

图3 ID\_和PROC\_INST\_ID\_不相等

上图中有两条数据，第二条数据（嵌入的子流程）的PARENT\_ID\_等于第一条数据的ID\_和PROC\_INST\_ID\_，并且两条数据的PROC\_INST\_ID\_相同。

上图还有一点特殊的地方，字段IS\_ACTIVE\_的值一个是0一个是1，说明正在执行子流程主流程挂起。

**3.Task**

刚刚说了ProcessInstance是和业务一对一关联的，和业务数据最亲密；Task是和用户最亲密的（UserTask），用户每天的待办事项就是一个个的Task对象。

从**图1**中看得出Execution和Task是一对多关系，Task可以是任何类型的Task实现，可以是用户任务（UserTask）、Java服务（ServiceTask）等，在实际流程运行中只不过面向对象不同，用户任务需要有人完成（complete），Java服务需要有系统自动执行（execution）。

图4 表ACT\_RU\_TASK

Task是在流程定义中**看到**的最大单位，每当一个task完成的（complete）时候会引擎把当前的任务移动到历史中，然后插入下一个任务插入到ACT\_RU\_TASK中。

从图4中可以看出

结合请假流程来说就是让用户点击“完成”按钮提交当前任务是的动作，引擎自动根据任务的顺序流或者排他分支判断走向。

**4.Activity**

Activity——活动。

图5 表ACT\_HI\_ACTINST

Activity包含了流程中所有的活动数据，例如开始事件（图5中的第1条）、各种分支（排他、并行等，图5中的第2条数据）、以及刚刚提到的Task执行记录（如图5中的第3、4条数据）。

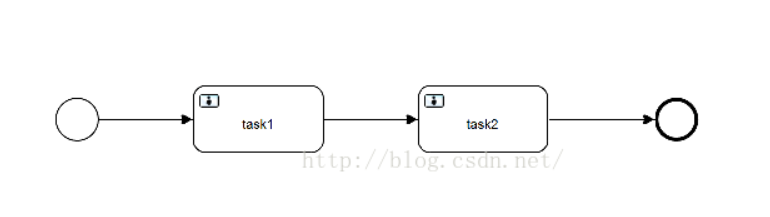
有些人认为Activity和Task是多对一关系，其实不是，从上图中可以看出来根本没有Task相关的字段。

结合请假流程来说如**Task**中提到的当完成流程的时候所有下一步要执行的任务（包括各种分支）都会创建一个Activity记录到数据库，例如领导审核节点点击“同意”按钮就会流转到人事审批节点，如果“驳回”那就流转到**调整请假内容**节点，每一次操作的task背后实际记录更详细的**活动**。

# Flowable启动流程到完成所有任务之间的数据库变化

来写一下Activiti 5.18版本从启动流程到整个流程结束之间数据库表的变化

先给出流程图，很简单的流程，就是两个UserTask：



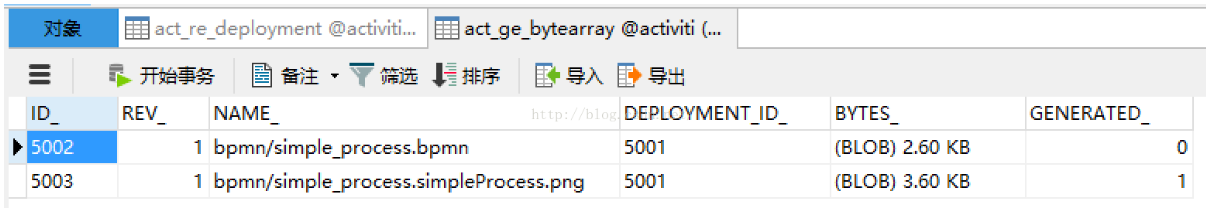
代码如下：

DeploymentBuilder builder=repositoryService.createDeployment();

Deployment deployment=builder.addClasspathResource("bpmn/simple\_process.bpmn").deploy();

以上代码是部署这个流程，部署完毕后，act\_re\_deployment表中会有一条部署记录，记录这次部署的基本信息，然后是act\_ge\_bytearray表中有两条记录，记录的是本次上传的bpmn文件和对应的图片文件，每条记录都有act\_re\_deployment表的外键关联，然后是act\_re\_procdef表中有一条记录，记录的是该bpmn文件包含的基本信息，包含act\_re\_deployment表外键。







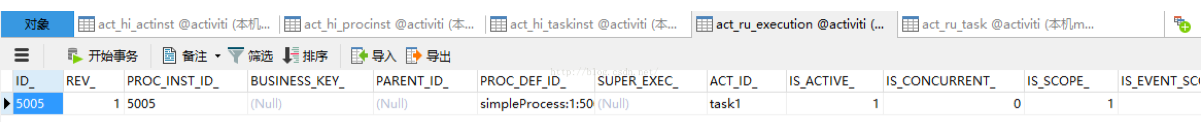
**注意看这个5001就是act\_re\_deployment表外键**

ProcessDefinition pd=repositoryService.createProcessDefinitionQuery().deploymentId(deployment.getId()).singleResult();

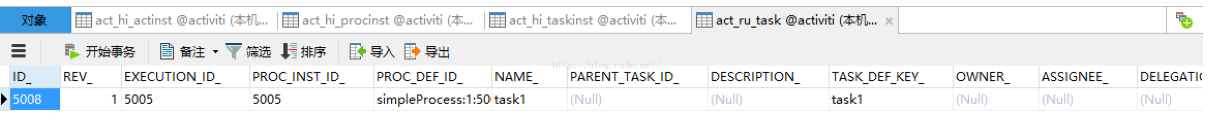
ProcessInstance pi=runtimeService.startProcessInstanceById(pd.getId());

以上代码是搜索此次部署对应的流程定义信息，也就是act\_re\_procdef表的信息查询。然后启动这个流程。

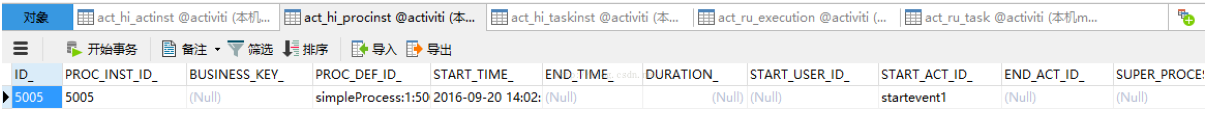
首先向act\_ru\_execution表中插入一条记录，记录的是这个流程定义的执行实例，其中id和proc\_inst\_id相同都是流程执行实例id，也就是本次执行这个流程定义的id，包含流程定义的id外键(simpleProcess:1:5004)。

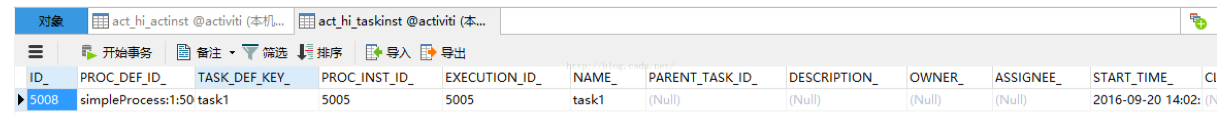


然后向act\_ru\_task插入一条记录，记录的是第一个任务的信息，也就是开始执行第一个任务。包括act\_ru\_execution表中的execution\_id外键和proc\_inst\_id外键，也就是本次执行实例id。



然后向act\_hi\_procinst表和act\_hi\_taskinst表中各插入一条记录，记录的是本次执行实例和任务的历史记录：





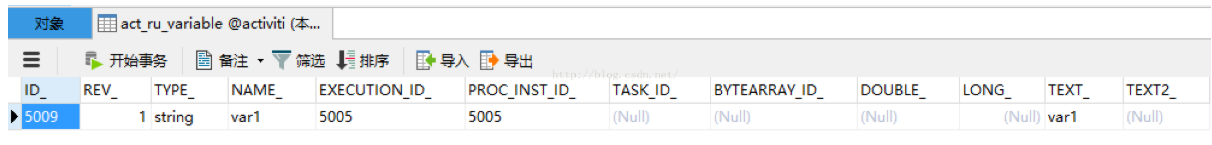
Task task1=taskService.createTaskQuery().processInstanceId(pi.getId()).taskDefinitionKey("task1").singleResult();

taskService.setVariable(task1.getId(), "var1", "var1");

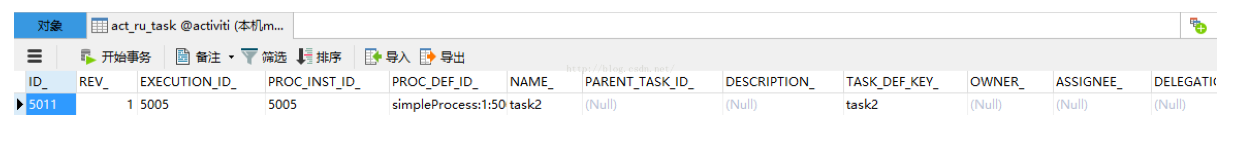
taskService.complete(task1.getId());

以上代码是查询流程本次执行实例下名为task1的任务，然后给任务设置全局变量，如果调用的是taskService.setVariableLocal方法，则任务执行完毕后，相关变量数据就会删除，然后再完成任务。

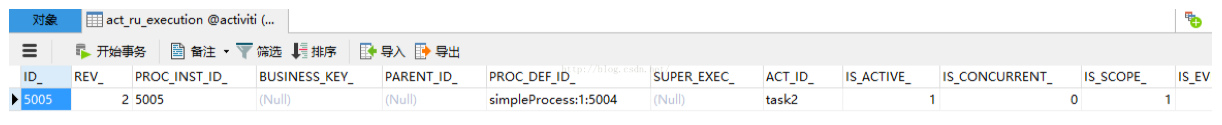
首先向act\_ru\_variable表中插入变量信息，包含本次流程执行实例的两个id外键，但不包括任务的id，因为setVariable方法设置的是全局变量，也就是整个流程都会有效的变量：



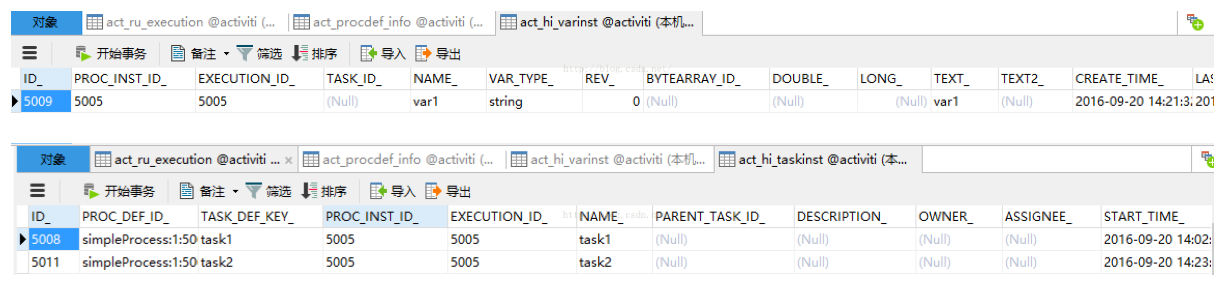
执行完task1后，act\_ru\_task表中task1的记录被删除，新插入task2的记录：



同时act\_ru\_execution表中的记录并没有删除，而是将正在执行的任务变成task2



同时向act\_hi\_var\_inst和act\_hi\_taskinst插入历史记录

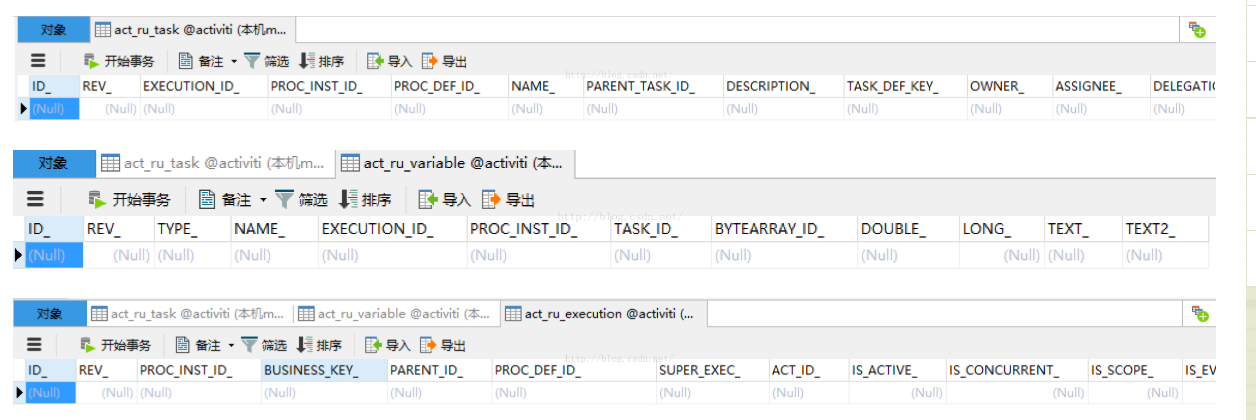


Task task2=taskService.createTaskQuery().processInstanceId(pi.getId()).taskDefinitionKey("task2").singleResult();

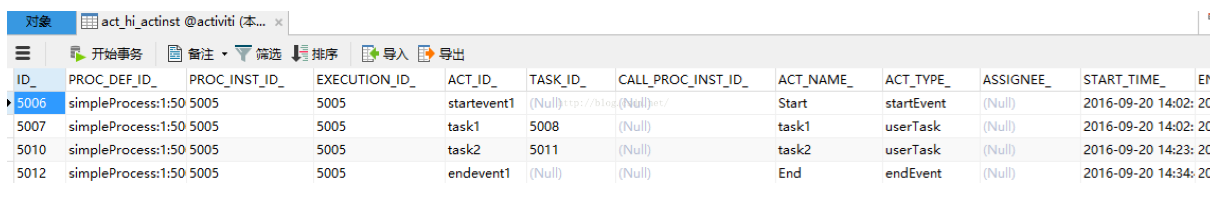
taskService.complete(task2.getId());

以上代码是查询本流程执行实例下的task2并完成task2。

此时整个流程执行完毕，act\_ru\_task，act\_ru\_execution和act\_ru\_variable表全被清空



其实全程有一个表一直在记录所有动作，就是act\_hi\_actinst表：



List<HistoricVariableInstance> vars=historyService.createHistoricVariableInstanceQuery().processInstanceId(pi.getId()).list();

for(HistoricVariableInstance var:vars){

System.out.println(var.getVariableName());

System.out.println(var.getValue());

}

以上这段代码是查询act\_hi\_varinst表中变量历史记录的。因为流程执行完毕act\_ru\_variable表被清空。

# flowable 删除数据库表先后顺序

工作流表删除顺序  
drop table act\_evt\_log;   
drop table act\_hi\_actinst;   
drop table act\_hi\_attachment;   
drop table act\_hi\_comment;   
drop table act\_hi\_detail;   
drop table act\_hi\_identitylink;   
drop table act\_hi\_procinst;   
drop table act\_hi\_taskinst;   
drop table act\_hi\_varinst;   
drop table act\_id\_info;   
drop table act\_id\_membership;   
drop table act\_id\_user;   
drop table act\_re\_model;   
drop table act\_ru\_event\_subscr;   
drop table act\_ru\_identitylink;   
DROP TABLE ACT\_RU\_DEADLETTER\_JOB;   
DROP TABLE ACT\_RU\_SUSPENDED\_JOB;   
DROP TABLE ACT\_RU\_TIMER\_JOB;   
drop table act\_ru\_job;   
drop table act\_ru\_task;   
drop table act\_id\_group;   
drop table act\_ru\_variable;   
drop table act\_ru\_execution;   
drop table ACT\_GE\_PROPERTY;   
drop table ACT\_PROCDEF\_INFO;   
drop table act\_ge\_bytearray;   
drop table act\_re\_deployment;   
drop table ACT\_RE\_PROCDEF;   
DROP SEQUENCE  ACT\_EVT\_LOG\_SEQ;

# flowable 核心表ACT\_RU\_EXECUTION 详解

一、ACT\_RU\_EXECUTION 表(很多初学者迷惑的一张表，以为是流程实例表，其实它叫执行实例表）：这个表和act\_run\_task表，一起控制了用户任务的产生与完成等。

 这个表是工作流程的核心表，这个表会体现主干与分支流程实例的概念，所以才有了执行实例这个叫法。

一般来讲一个流程实例都有一条主线。如果流程为直线流程，那么流程实例在这个表中只有一条记录对应。

但如果流程有会签多实例时，以及并行网关时，这时候它就有流程实例和执行实例，一对多的关系，所以一定要理解流实例和执行实例的区别，

不要把它执行实例误以为流程实例表，当在并行网关和会签多实例时，它是会产生多个执行实例，刚创建时各个执行实例的IS\_ACTIVE\_这个字段的值都是为1，即激活状态

当每完成一个执行实例时，它会把IS\_ACTIVE设为0，非激活状态，当所有执行实例完成后，它才会转移到历史，把这个多实例自动删除。

另外当有子流程，子流程的实例是处于激活状态，而主干流程的实例处于非激活状态。

 \*\*ID\_\*\*：EXECUTION主键，这个主键有可能和PROC\_INST\_ID\_相同，相同的情况表示这条记录为主实例记录。

\*\*REV\_\*\*：表示数据库表更新次数。

\*\*PROC\_INST\_ID\_\*\*：一个流程实例不管有多少条分支实例，这个ID都是一致的。

 \*\*BUSINESS\_KEY\_\*\*:这个为业务主键，主流程才会使用业务主键，另外这个业务主键字段在表中有唯一约束。

\*\*PARENT\_ID\_\*\*：这个记录表示父实例ID，如上图，同步节点会产生两条执行记录，这两条记录的父ID为主线的ID。

\*\*PROC\_DEF\_ID\_ :\*\*流程定义ID

\*\*SUPER\_EXEC\*\* ： 这个如果存在表示这个实例记录为一个外部子流程记录，对应主流程的主键ID。

 \*\*ACT\_ID\_\*\*：表示流程运行到的节点，如上图主实例运行到ANDGateway1 节点。

两个子实例运行到UserTask1,UserTask2节点。

 \*\*IS\_ACTIVE\_ : \*\*是否活动流程实例，比如上图，主流程为非活动实例，下面两个为活动实例，如果UserTask2完成，那么这个值将变为0即非活动。

\*\*IS\_CONCURRENT\_:\*\*是否并发。上图同步节点后为并发，如果是并发多实例也是为1。

\*\*IS\_SCOPE\_\*\*: 这个字段我跟踪了一下不同的流程实例，如会签，子流程，同步等情况，发现主实例的情况这个字段为1，子实例这个字段为0。

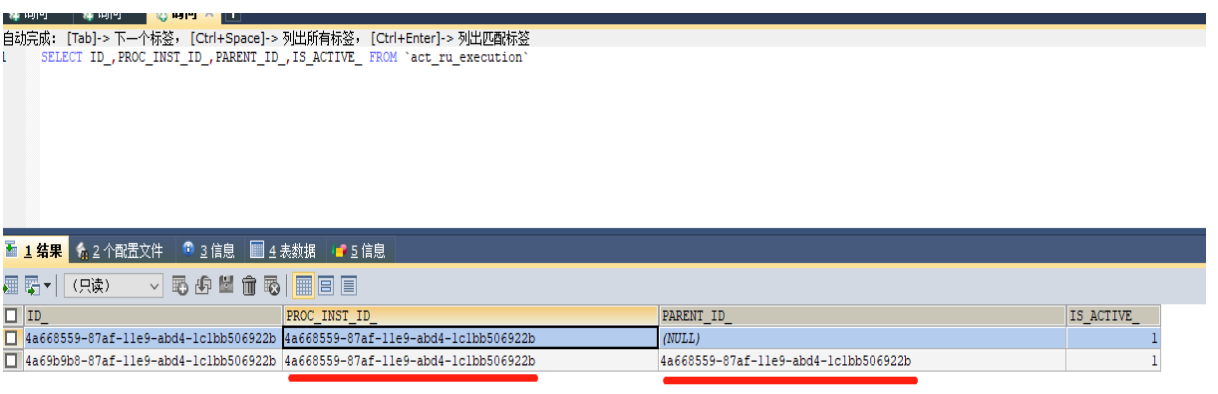
 \*\*TENANT\_ID\_ :\*\*  这个字段表示租户ID。可以应对多租户的设计。

\*\*IS\_EVENT\_SCOPE\*\*: 没有使用到事件的情况下，一般都为0。

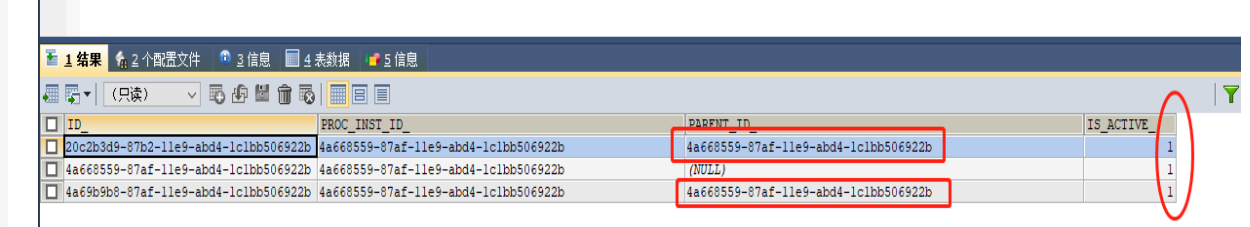
\*\*SUSPENSION\_STATE\_\*\*： 这个表示是否暂停。

二、直线串行流程时，当前执行实例数据库截图：

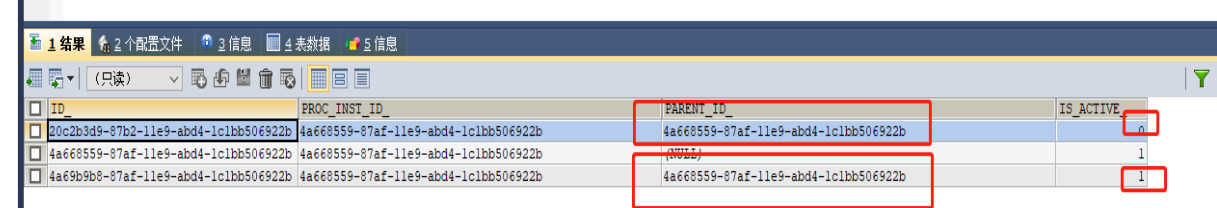
1.启动流程，即创建流程实例时，缺省为创建2条记录，其中一条是开始事件的执行实例，这个一直存在，只到流程结束后才会自动删除，另一条是普通的节点，所以它的流程实例id和PARENT\_ID\_字段即主干执行实例id字段值是一样的，完成任务时，它会第二转入历史任务和历史活动表中，如果创建的下一个节点还是普通节点，则它的第二条记录会变化。



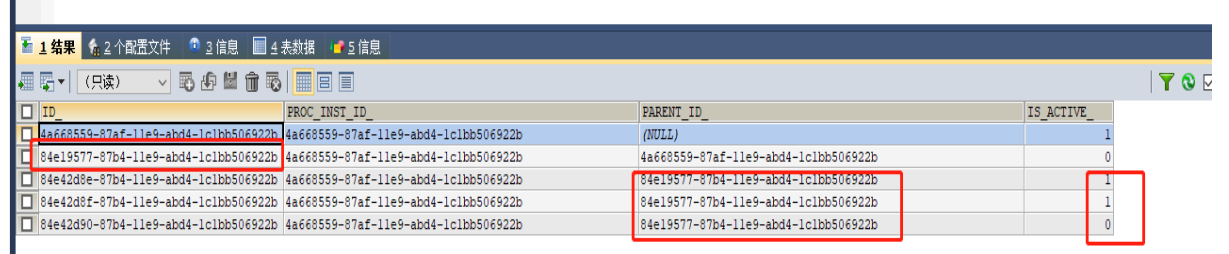
2.并行网关时：创建了2个分支，他们的parant\_id和流程实例id(proc\_inst\_id字段)都一样，缺省is\_active都是1表示激活状态，当完成其中一个分支节点时，is\_active就会变成0，等候其它节点一起完成后，才会转到历史，到时删除这些执行实例。并行网关的执行实例，它们的活动节点key是不一样，act\_run\_task中的TASK\_DEF\_ID\_字段值是不一样（区别于会签多实例，会签多实例这个值是一样的）。



完成了其中一条分支节点时，is\_active变为了0。



3.会签多实例时，如下是一个3人的会签，它会产生5条记录，一条是开始事件执行实例，一条是主干执行实例，只是它的状态为非激活状态，其它3个会签执行实例的parant\_id值均是主干执行实例的id值，其中一个人做完了，这时候的act\_ru\_execution数据还是3条，但完成任务的那个执行实例is\_active由1变为0了，它也一样等待其它2人会签均完成了才会转移到历史，如果查看act\_ru\_task表，则会发现原来3条，变为2条了，其中一条它已经转移到历史用户任务了。



flowable的办理方式，分为两种：签收模式办理和直接办理。

术语：

Assignee: 任务的受理人，即执行人。它有两种情况（有值,NULL)

Owner: 任务的委托人。

CandidateGroup: 候选用户组

CandidateUser: 候选人

delegateTask: 委派任务/签收的任务

resolveTask:  委派任务的代办，任务的拥有者把任务委派他人来办理，他人办完后，又重新回到任务拥有者，会产生流转记录。

turnTask： 转办任务，只是改变当前任务的办理人而已，不会产生流转记录。

CompleteTask: 完成任务，或叫办结提交下一步。

claimTask：任务签收

一、签收后办理模式

  任务创建后，流程进入一个等待状态，需要用户去签收任务，即接收任务的一个过程。原理就是它的执行实例表act\_ru\_execution创建了一条记录，但act\_ru\_task表中是没有创建这个记录，只有签收后act\_ru\_task才会生成一条任务记录。签收办理人，可以分为：候选人（一人或者多人，之间逗号分开）和候选用户组（一个或者多个组，之间逗号分开）。

   这种模式就是任务的抢占模式，谁先签收，这个任务就归谁。真实的任务其实只有一条，只是还没有在act\_ru\_task生成。

二、直接办理模式

   任务创建后，直接给了提交到下一个节点，给下一个节点创建任务时，就指定了这个任务的受理人（Assignee\_)

三、直接办理任务三种指派人方式：

 (1)xml流程模型中直接设置固定死的用户id或者用户账号，例如：

      <userTask id="usertask1" name="审批"  flowable:assignee="userId2"></userTask>

      也可以配置成候选人或者候选组方式。

 (2)任务节点上设置流程变量，这个流程变量的值由上一个节点来设置，例如：

      <userTask id="usertask1" name="审批" flowable:assignee="#{userId}"></userTask>

 (3)自定义一个监听类，实现TaskListener接口，细节查看如何定义监听器类及配置到流程模型xml中相关章节。

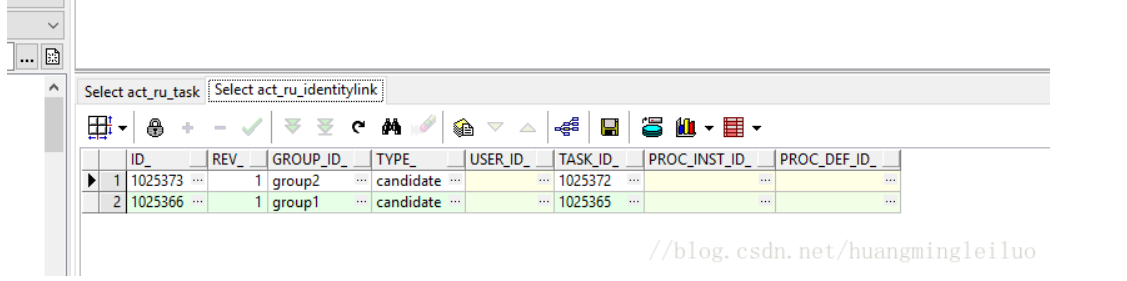
四、候选人和候选组的配置：

<userTask id="usertask2" name="部门经理审批" flowable:candidateGroups="group1,group2"></userTask>

<userTask id="usertask2" name="部门经理审批" flowable:candidateUsers="user1,user2"></userTask>

有一张数据库表和它有关：

act\_ru\_identitylink,这个表里有几个字段要理解一下：group\_id（候选组时有值），User\_id（候选人模式有值 ）,Type(指明是候选人还是候选组），如下：



# 为了启动流程实例时给流程加一个标题

我们经常希望待办任务列表有一列是流程实例的名称，即标题，例如，我们希望流程的标题是：张三 2019-12-04 16：40：20 的请假申请，通常会采用一个命名规则：发起人+发起时间+流程模型名称把它作为流程的标题。这时候就希望启动流程实例时可以设置流程实例的名称，方法如下： 通过流程实例构造器来启动流程实例。

/\*\*

\* @Description: 方法二：通过ProcessInstanceBuilder来启动创建流程实例

\* 常用在要为流程实例设置标题时用

\*

\* @Date: 2019/12/4 22:33

\* @Param:

\* @Param:

\* @return:

\*\*/

public void StartProcInstanceByBuilder(){

// 获取流程构造器

ProcessInstanceBuilder processInstanceBuilder = runtimeService.createProcessInstanceBuilder();

// 设置参数

processInstanceBuilder.name("钟招坤 2019-12-04 请假"); // 流程实例标题

processInstanceBuilder.businessKey("appxx"); // 业务key

processInstanceBuilder.processDefinitionId("apply:001"); //流程定义key

// 启动（即创建）流程实例

ProcessInstance instance = processInstanceBuilder.start();

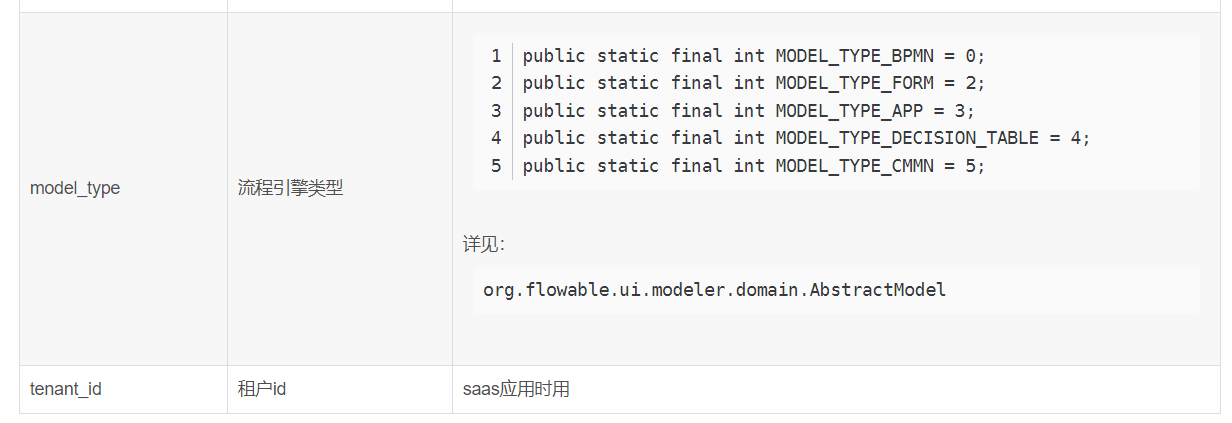
System.out.println(instance.getId());

}

# 关于流程设计器：act\_de\_mode的表详解

flowable从6.4.1开始，act\_de\_model代替原来的act\_re\_model表，act\_re\_model将作废，大家要注意。





# 关于流程设计器： bpmn.js与vue的整合

一、前言：

       由于flowable本身带的Moder风格实在与前端页面风格不一样，同时由于Modeler采用Angular.js写的，改造起来相对有一定的难度，所以打算换成bpmn.js当成流程设计器。

二、与vue的整合

1.安装以下包进开发环境

 npm install --save    bpmn-js;

npm install -- save bpmn-js-properties-panel;

npm install --save camunda-bpmn-moddle;

2.定制开发以下功能：

  （1）打开：打开本地的\*.bpmn20.xml模型文件,并在bpmn.js插件的画布上显示出来。

   （2）创建：创建一个新的流程，在画布上供用户拖拉完成。

   （3）导出流程模板：导成一个xml或者zip的形式。

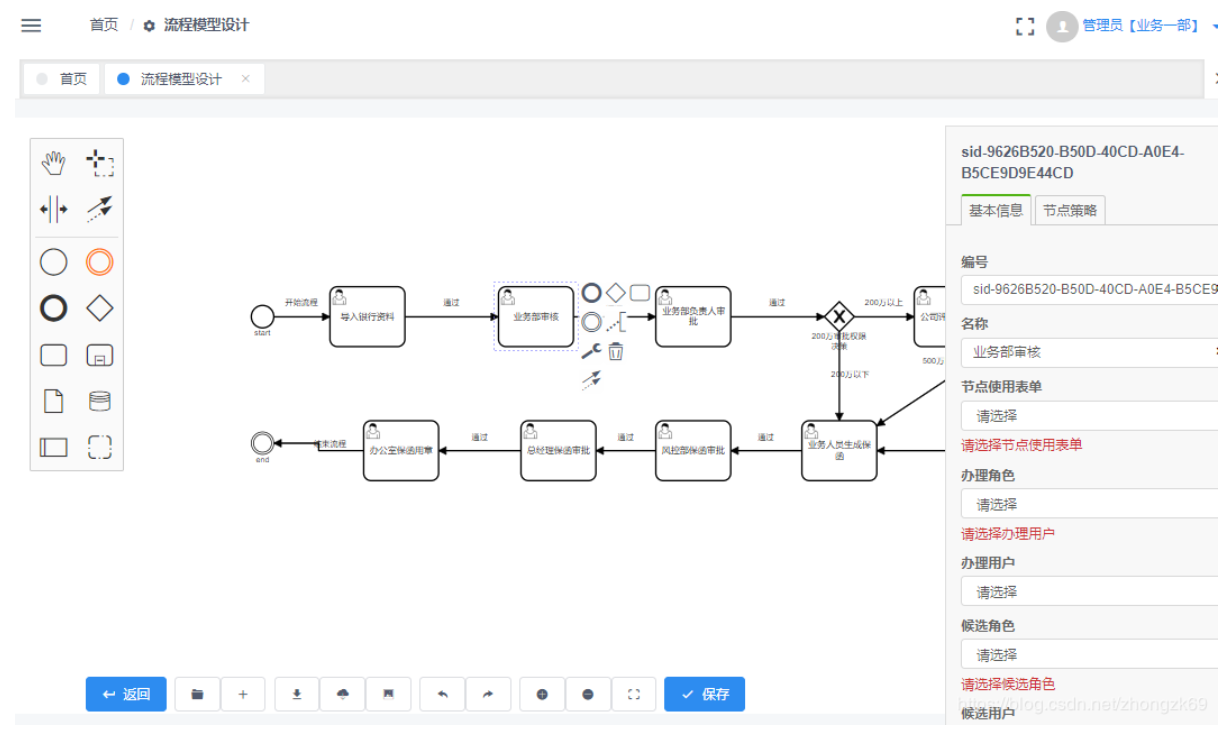
   （4）撤销：支持向前或向后撤销，即撤销在画布上刚才操作。

     (5)  放大/缩小：支持画布的放大与缩小，以及重置恢常正常大小。

     (6) 保存流程的模型到自己的定义数据库表，同时同步更新到act\_de\_model表中。

     (7) 节点属性面板的定制功能开发。

三、整合的真实效果如下：

四、功能代码参

考bpmn.js官方例子代码即可。

# 关于流程设计器： bpmn.js 彻底汉化

一、bpmn.js官方例子有一个bpmn-js-examples/i18n专门讲了如何将bpmn-js中显示的字符转换为另一种语言的方案，本文根据官方的描述进行的实战。

二、原理：

bpmn-js在实例化时，它把转换语言的功能（custom-translate）作为一个额外的配置模块，这样就覆盖了缺省的原来语言，在你

BpmnModeler实例化的js文件中，加入以下代码：

var customTranslate = {

translate: [ 'value', require('./custom-translate/custom-translate') ]

};

var modeler = new BpmnModeler({

// ...

additionalModules: [

customTranslate

]

});

三、实现步骤：

（1）定义customTranslate.js:  注意红字部分，官方没有的，官方给的汉化包不完整。

import translations from './translations';

export default function customTranslate(template, replacements) {

  replacements = replacements || {};

  // Translate

  template = translations[template] || template;

  // Replace

  return template.replace(/{([^}]+)}/g, function(\_, key) {

    return replacements[key] || '{' + key + '}';

  });

}

  (2) 定义转换的英文与中文对应的translations.js ，

具体更详细，它的官方提供了不同国家的语言包提供下载：https://github.com/bpmn-io/bpmn-js-i18n

export default {

// Labels

'Activate the global connect tool': '激活全局连接工具',

'Append {type}': '添加 {type}',

'Add Lane above': '在上面添加道',

'Divide into two Lanes': '分割成两个道',

'Divide into three Lanes': '分割成三个道',

'Add Lane below': '在下面添加道',

'Append compensation activity': '追加补偿活动',

'Change type': '修改类型',

'Connect using Association': '使用关联连接',

'Connect using Sequence/MessageFlow or Association': '使用顺序/消息流或者关联连接',

'Connect using DataInputAssociation': '使用数据输入关联连接',

Remove: '移除',

'Activate the hand tool': '激活抓手工具',

'Activate the lasso tool': '激活套索工具',

'Activate the create/remove space tool': '激活创建/删除空间工具',

'Create expanded SubProcess': '创建扩展子过程',

'Create IntermediateThrowEvent/BoundaryEvent': '创建中间抛出事件/边界事件',

'Create Pool/Participant': '创建池/参与者',

'Parallel Multi Instance': '并行多重事件',

'Sequential Multi Instance': '时序多重事件',

DataObjectReference: '数据对象参考',

DataStoreReference: '数据存储参考',

Loop: '循环',

'Ad-hoc': '即席',

'Create {type}': '创建 {type}',

Task: '任务',

'Send Task': '发送任务',

'Receive Task': '接收任务',

'User Task': '用户任务',

'Manual Task': '手工任务',

'Business Rule Task': '业务规则任务',

'Service Task': '服务任务',

'Script Task': '脚本任务',

'Call Activity': '调用活动',

'Sub Process (collapsed)': '子流程（折叠的）',

'Sub Process (expanded)': '子流程（展开的）',

'Start Event': '开始事件',

StartEvent: '开始事件',

'Intermediate Throw Event': '中间事件',

'End Event': '结束事件',

EndEvent: '结束事件',

'Create Gateway': '创建网关',

'Create Intermediate/Boundary Event': '创建中间/边界事件',

'Message Start Event': '消息开始事件',

'Timer Start Event': '定时开始事件',

'Conditional Start Event': '条件开始事件',

'Signal Start Event': '信号开始事件',

'Error Start Event': '错误开始事件',

'Escalation Start Event': '升级开始事件',

'Compensation Start Event': '补偿开始事件',

'Message Start Event (non-interrupting)': '消息开始事件（非中断）',

'Timer Start Event (non-interrupting)': '定时开始事件（非中断）',

'Conditional Start Event (non-interrupting)': '条件开始事件（非中断）',

'Signal Start Event (non-interrupting)': '信号开始事件（非中断）',

'Escalation Start Event (non-interrupting)': '升级开始事件（非中断）',

'Message Intermediate Catch Event': '消息中间捕获事件',

'Message Intermediate Throw Event': '消息中间抛出事件',

'Timer Intermediate Catch Event': '定时中间捕获事件',

'Escalation Intermediate Throw Event': '升级中间抛出事件',

'Conditional Intermediate Catch Event': '条件中间捕获事件',

'Link Intermediate Catch Event': '链接中间捕获事件',

'Link Intermediate Throw Event': '链接中间抛出事件',

'Compensation Intermediate Throw Event': '补偿中间抛出事件',

'Signal Intermediate Catch Event': '信号中间捕获事件',

'Signal Intermediate Throw Event': '信号中间抛出事件',

'Message End Event': '消息结束事件',

'Escalation End Event': '定时结束事件',

'Error End Event': '错误结束事件',

'Cancel End Event': '取消结束事件',

'Compensation End Event': '补偿结束事件',

'Signal End Event': '信号结束事件',

'Terminate End Event': '终止结束事件',

'Message Boundary Event': '消息边界事件',

'Message Boundary Event (non-interrupting)': '消息边界事件（非中断）',

'Timer Boundary Event': '定时边界事件',

'Timer Boundary Event (non-interrupting)': '定时边界事件（非中断）',

'Escalation Boundary Event': '升级边界事件',

'Escalation Boundary Event (non-interrupting)': '升级边界事件（非中断）',

'Conditional Boundary Event': '条件边界事件',

'Conditional Boundary Event (non-interrupting)': '条件边界事件（非中断）',

'Error Boundary Event': '错误边界事件',

'Cancel Boundary Event': '取消边界事件',

'Signal Boundary Event': '信号边界事件',

'Signal Boundary Event (non-interrupting)': '信号边界事件（非中断）',

'Compensation Boundary Event': '补偿边界事件',

'Exclusive Gateway': '互斥网关',

'Parallel Gateway': '并行网关',

'Inclusive Gateway': '相容网关',

'Complex Gateway': '复杂网关',

'Event based Gateway': '事件网关',

Transaction: '转运',

'Sub Process': '子流程',

'Event Sub Process': '事件子流程',

'Collapsed Pool': '折叠池',

'Expanded Pool': '展开池',

// Errors

'no parent for {element} in {parent}': '在{parent}里，{element}没有父类',

'no shape type specified': '没有指定的形状类型',

'flow elements must be children of pools/participants': '流元素必须是池/参与者的子类',

'out of bounds release': 'out of bounds release',

'more than {count} child lanes': '子道大于{count} ',

'element required': '元素不能为空',

'diagram not part of bpmn:Definitions': '流程图不符合bpmn规范',

'no diagram to display': '没有可展示的流程图',

'no process or collaboration to display': '没有可展示的流程/协作',

'element {element} referenced by {referenced}#{property} not yet drawn':

'由{referenced}#{property}引用的{element}元素仍未绘制',

'already rendered {element}': '{element} 已被渲染',

'failed to import {element}': '导入{element}失败',

// 属性面板的参数

Id: '编号',

Name: '名称',

General: '常规',

Details: '详情',

'Message Name': '消息名称',

Message: '消息',

Initiator: '创建者',

'Asynchronous Continuations': '持续异步',

'Asynchronous Before': '异步前',

'Asynchronous After': '异步后',

'Job Configuration': '工作配置',

Exclusive: '排除',

'Job Priority': '工作优先级',

'Retry Time Cycle': '重试时间周期',

Documentation: '文档',

'Element Documentation': '元素文档',

'History Configuration': '历史配置',

'History Time To Live': '历史的生存时间',

Forms: '表单',

'Form Key': '表单key',

'Form Fields': '表单字段',

'Business Key': '业务key',

'Form Field': '表单字段',

ID: '编号',

Type: '类型',

Label: '名称',

'Default Value': '默认值',

Validation: '校验',

'Add Constraint': '添加约束',

Config: '配置',

Properties: '属性',

'Add Property': '添加属性',

Value: '值',

Listeners: '监听器',

'Execution Listener': '执行监听',

'Event Type': '事件类型',

'Listener Type': '监听器类型',

'Java Class': 'Java类',

Expression: '表达式',

'Must provide a value': '必须提供一个值',

'Delegate Expression': '代理表达式',

Script: '脚本',

'Script Format': '脚本格式',

'Script Type': '脚本类型',

'Inline Script': '内联脚本',

'External Script': '外部脚本',

Resource: '资源',

'Field Injection': '字段注入',

Extensions: '扩展',

'Input/Output': '输入/输出',

'Input Parameters': '输入参数',

'Output Parameters': '输出参数',

Parameters: '参数',

'Output Parameter': '输出参数',

'Timer Definition Type': '定时器定义类型',

'Timer Definition': '定时器定义',

Date: '日期',

Duration: '持续',

Cycle: '循环',

Signal: '信号',

'Signal Name': '信号名称',

Escalation: '升级',

Error: '错误',

'Link Name': '链接名称',

Condition: '条件名称',

'Variable Name': '变量名称',

'Variable Event': '变量事件',

'Specify more than one variable change event as a comma separated list.':

'多个变量事件以逗号隔开',

'Wait for Completion': '等待完成',

'Activity Ref': '活动参考',

'Version Tag': '版本标签',

Executable: '可执行文件',

'External Task Configuration': '扩展任务配置',

'Task Priority': '任务优先级',

External: '外部',

Connector: '连接器',

'Must configure Connector': '必须配置连接器',

'Connector Id': '连接器编号',

Implementation: '实现方式',

'Field Injections': '字段注入',

Fields: '字段',

'Result Variable': '结果变量',

Topic: '主题',

'Configure Connector': '配置连接器',

'Input Parameter': '输入参数',

Assignee: '代理人',

'Candidate Users': '候选用户',

'Candidate Groups': '候选组',

'Due Date': '到期时间',

'Follow Up Date': '跟踪日期',

'Specify more than one group as a comma separated list.': '多个用户使用逗号隔开',

Priority: '优先级',

// eslint-disable-next-line no-template-curly-in-string

'The follow up date as an EL expression (e.g. ${someDate} or an ISO date (e.g. 2015-06-26T09:54:00)':

'跟踪日期必须符合EL表达式，如： ${someDate} ,或者一个ISO标准日期，如：2015-06-26T09:54:00',

// eslint-disable-next-line no-template-curly-in-string

'The due date as an EL expression (e.g. ${someDate} or an ISO date (e.g. 2015-06-26T09:54:00)':

'跟踪日期必须符合EL表达式，如： ${someDate} ,或者一个ISO标准日期，如：2015-06-26T09:54:00',

Variables: '变量',

'Candidate Starter Users': '选择启动候选人',

'Candidate Starter Configuration': '候选人启动器配置',

'Candidate Starter Groups': '候选人启动组',

'This maps to the process definition key.': '编号将映射到流程主键.',

save: '保存',

Tools: '工具',

FlowGateway: '流程网关',

ProcessControl: '流程节点',

'Create StartEvent': '开始节点',

'Create EndEvent': '结束节点',

'Create ExclusiveGateway': '互斥网关',

'Create ParallelGateway': '并行网关',

'Create Task': '任务节点',

'Create UserTask': '用户任务节点',

'Condition Type': '条件类型',

// 左侧工具箱补充汉化项 热水2020.1.12

'Create Group': '创建组',

'Create DataObjectReference': '创建数据对象引用',

'Create DataStoreReference': '创建数据存储引用',

// 节点添加Pad 补充汉化 热水2020.1.12

'Append EndEvent': '追加结束事件节点',

'Append Gateway': '追加网关节点',

'Append UserTask': '追加用户任务节点',

'Append Intermediate/Boundary Event': '追加中间或边界事件',

'Append TextAnnotation': '追加文本批注' // 此句要有效，必须在CustomContexPadProvide给此节点增加一个translate('Append TextAnnotation')

}

特别注意在自定义的CustomContextPadProvider.js（由于缺省是task，要改为缺省为UserTask，所以必须自定义ContextPad）中，对备注的汉化，如下：

assign(actions, {

'append.text-annotation': appendAction(

'bpmn:TextAnnotation',

'bpmn-icon-text-annotation',

translate('Append TextAnnotation')

),

（3）BpmnModeler实例化的js文件中，加入以下代码：

import customTranslate from './customTranslate/customTranslate';

var customTranslate = {

translate: [ 'value', require('./custom-translate/custom-translate') ]

};

var modeler = new BpmnModeler({

// ...

additionalModules: [

customTranslate

]

});

(3)成功截图：



注意：还是补充一些汉化项，官方下载的也不太规范，自己要修正或者补充一些。以上做完，就彻底完成了汉化。