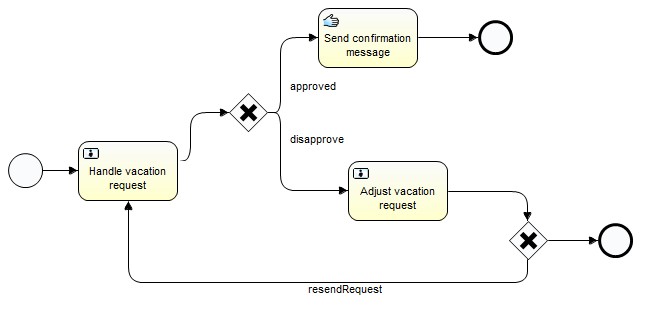
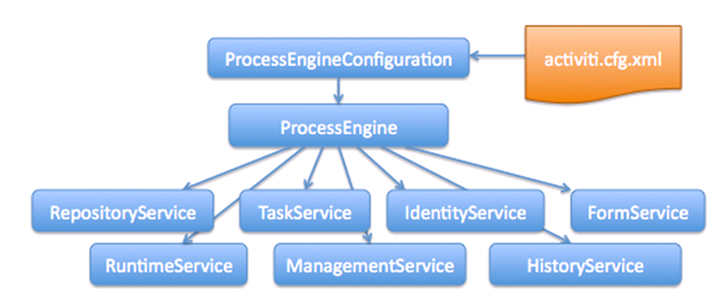
# 工作流

## 一 Activiti介绍

Activiti5是由Alfresco软件在2010年5月17日发布的业务流程管理（BPM）框架，它是覆盖了业务流程管理、工作流、服务协作等领域的一个开源的、灵活的、易扩展的可执行流程语言框架。Activiti基于Apache许可的开源BPM平台，创始人Tom Baeyens是JBoss jBPM的项目架构师，它特色是提供了eclipse插件，开发人员可以通过插件直接绘画出业务流程图。ProcessEngine（工作流引擎）对象，这是Activiti工作的核心。负责生成流程运行时的各种实例及数据、监控和管理流程的运行。业务流程建模与标注（Business Process Model and Notation，BPMN) ，描述流程的基本符号，包括这些图元如何组合成一个业务流程图（Business Process Diagram）



## 二 Activiti核心API结构



## 三 Activiti的数据表结构

Activiti的后台是有数据库的支持，所有的表都以ACT\_开头。 第二部分是表示表的用途的两个字母标识。 用途也和服务的API对应。

ACT\_RE\_\*: 'RE'表示repository。 这个前缀的表包含了流程定义和流程静态资源 （图片，规则，等等）。

ACT\_RU\_\*: 'RU'表示runtime。 这些运行时的表，包含流程实例，任务，变量，异步任务，等运行中的数据。 Activiti只在流程实例执行过程中保存这些数据， 在流程结束时就会删除这些记录。 这样运行时表可以一直很小速度很快。

ACT\_ID\_\*: 'ID'表示identity。 这些表包含身份信息，比如用户，组等等。

ACT\_HI\_\*: 'HI'表示history。 这些表包含历史数据，比如历史流程实例， 变量，任务等等。

ACT\_GE\_\*: 通用数据， 用于不同场景下，如存放资源文件。

表结构操作：

### 3.1：资源库流程规则表

act\_re\_deployment 部署信息表

act\_re\_model 流程设计模型部署表

act\_re\_procdef 流程定义数据表

### 3.2：运行时数据库表

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的相关信息

act\_ru\_task 运行时任务节点表

act\_ru\_variable 运行时流程变量数据表

### 3.3：历史数据库表

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\_identitylink 历史流程人员表

act\_hi\_detail 历史详情表，提供历史变量的查询

act\_hi\_procinst 历史流程实例表

act\_hi\_taskinst 历史任务实例表

act\_hi\_varinst 历史变量表

### 3.4：组织机构表

act\_id\_group 用户组信息表

act\_id\_info 用户扩展信息表

act\_id\_membership 用户与用户组对应信息表

act\_id\_user 用户信息表

这四张表很常见，基本的组织机构管理，关于用户认证方面建议还是自己开发一套，组件自带的功能太简单，使用中有很多需求难以满足

### 3.5：通用数据表

act\_ge\_bytearray 二进制数据表

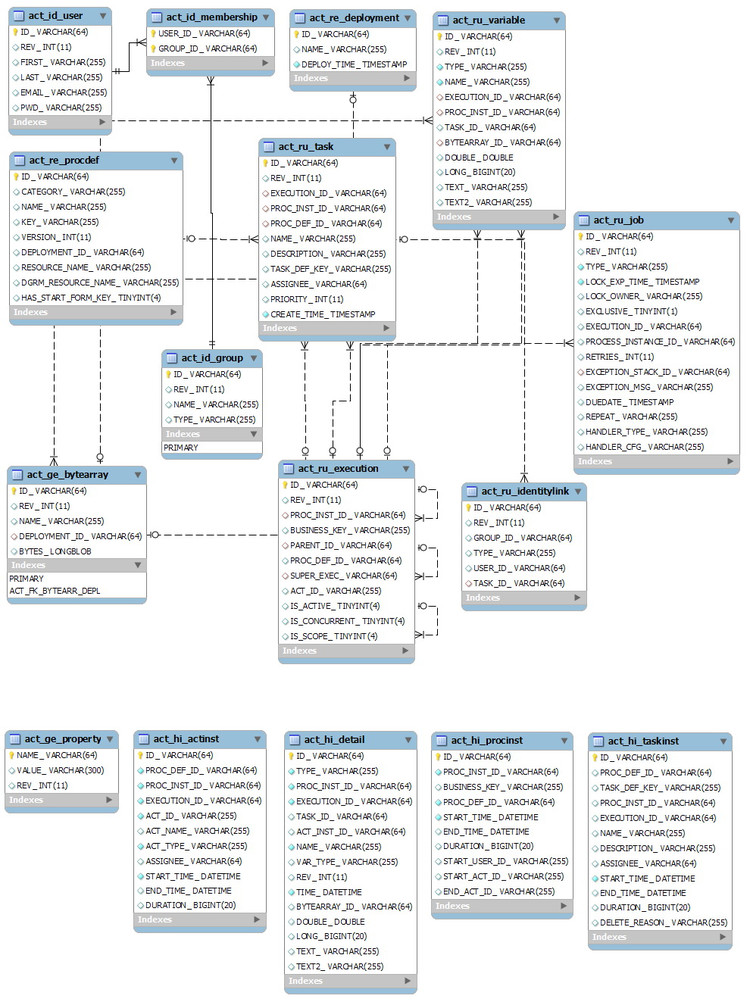
act\_ge\_property 属性数据表存储整个流程引擎级别的数据,初始化表结构时，会默认插入三条记录，

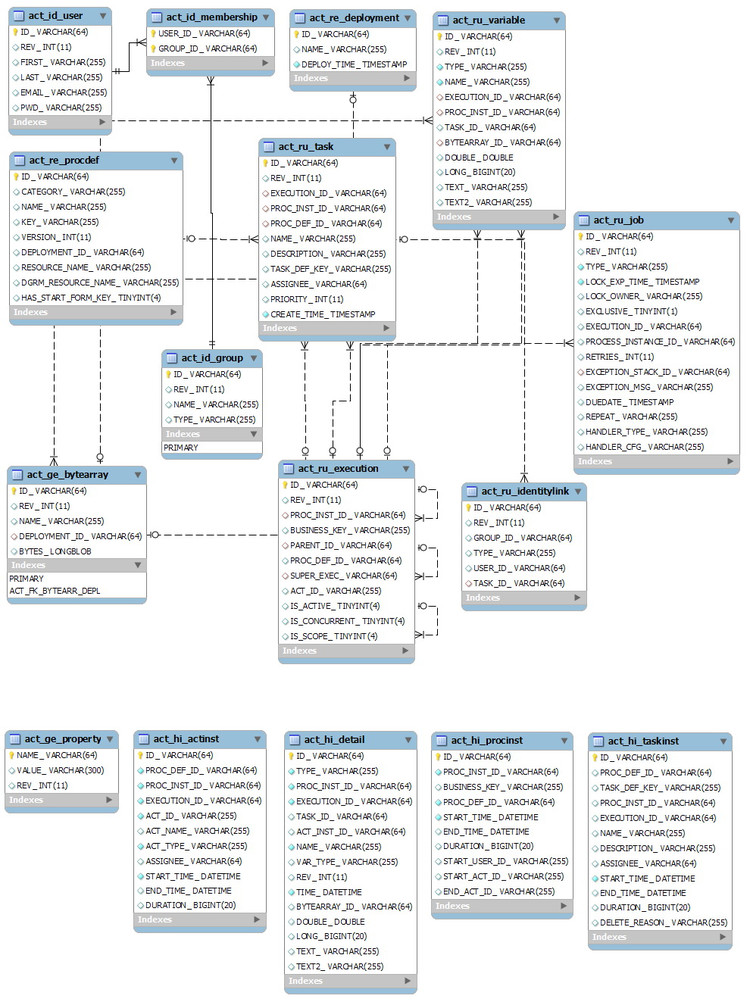
## 四 Activiti 表的关系

### 4.1 表结构模型

****

### 4.2 E-R图





## 五 Activiti的使用过程

### 5.1 流程

#### 5.1.1 自动建表(代码)

processEngineConfiguration=ProcessEngineConfiguration.*createStandaloneInMemProcessEngineConfiguration*();

processEngineConfiguration.setJdbcDriver("com.mysql.jdbc.Driver");

processEngineConfiguration.setJdbcUrl("jdbc:mysql://localhost/testactivity?useUnicode=true&characterEncoding=gbk");

processEngineConfiguration.setJdbcUsername("root");

processEngineConfiguration.setJdbcPassword("root");

//如果没有表就自动创建processEngineConfiguration.setDatabaseSchemaUpdate(ProcessEngineConfiguration.***DB\_SCHEMA\_UPDATE\_TRUE***);

//创建流程引擎

ProcessEngine buildProcessEngine = processEngineConfiguration.buildProcessEngine();

#### 5.1.2 自动建表（配置文件）

在类路径下放置activiti.cfg.xml文件。配置如下

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:tx=*"http://www.springframework.org/schema/tx"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-2.5.xsd*

*http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd"*>

<bean id=*"processEngineConfiguration"* class=*" org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration"*>

<!-- 配置数据库连接 -->

<property name=*"jdbcDriver"* value=*"com.mysql.jdbc.Driver"*></property>

<property name=*"jdbcUrl"* value=*"jdbc:mysql://localhost/testactivity?useUnicode=true&amp;characterEncoding=gbk"*></property>

<property name=*"jdbcUsername"* value=*"root"*></property>

<property name=*"jdbcPassword"* value=*"root"*></property

<!-- 建表策略 没有表,自动创建-->

<property name=*"databaseSchemaUpdate"* value=*"true"*></property>

</bean>

</beans>

荣国如下所示的代码加载配置文件

ProcessEngineConfiguration createProcessEngineConfigurationFromResource = ProcessEngineConfiguration.*createProcessEngineConfigurationFromResource*("activiti.cfg.xml");

ProcessEngine buildProcessEngine =createProcessEngineConfigurationFromResource.buildProcessEngine();

#### 5.1.3 流程部署(xml方式)

**public** **void** deploy(){

RepositoryService repositoryService = buildProcessEngine.getRepositoryService();

DeploymentBuilder createDeployment = repositoryService.createDeployment();

createDeployment.addClasspathResource("diagrams/MyProcess.bpmn");

createDeployment.addClasspathResource("diagrams/MyProcess.png");

createDeployment.name("请假流程");//部署名称

createDeployment.category("审批流程");//流程分类

Deployment deploy = createDeployment.deploy();

System.***out***.println(deploy.getId());

}

SELECT \* FROM act\_re\_deployment;//部署表

SELECT \* FROM act\_re\_procdef;//流程定义表

SELECT \* FROM act\_ge\_property;//通用属性表

SELECT \* FROM act\_ge\_bytearray;//图片存放表

#### 5.1.4 流程部署(zip文件方式)

注意打包的方式，不要压缩成了.rar的包，导致部署成功了，

但是表里面没数据，也不会提示错误

**public** **void** deploy() **throws** FileNotFoundException{

RepositoryService repositoryService = processEngine.getRepositoryService();

InputStream resourceAsStream = getClass().getClassLoader().getResourceAsStream("Buyprocess.zip");

ZipInputStream inputStream=**new** ZipInputStream(resourceAsStream);

Deployment deploy = repositoryService.createDeployment()

.addZipInputStream(inputStream)

.name("药品采购流程")

.category("采购流程")

.deploy();

System.***out***.println("部署id :"+deploy.getId());

System.***out***.println("部署名称 :"+deploy.getName());

}

#### 5.1.5 查询流程定义

**public** **void** findProcessDefinitionId(){

List<ProcessDefinition> list = processEngine.getRepositoryService().createProcessDefinitionQuery()

//.processDefinitionId(arg0)

.processDefinitionKey("Buyprocess")

//.processDefinitionVersion(arg0)

//.processDefinitionName(arg0)

//.latestVersion()

.list();

**for**(ProcessDefinition pro:list){

System.***out***.println("流程定义id:"+pro.getId());

System.***out***.println("流程定义key:"+pro.getKey());

System.***out***.println("流程定义name:"+pro.getName());

System.***out***.println("流程定义version:"+pro.getVersion());

}

}

#### 5.1.6 获取流程部署图片

**public** **void** viemimg() **throws** IOException{

String deploymentId="1";

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

String imgName=**null**;

**for**(String fileName:deploymentResourceNames){

**if**(fileName.indexOf(".png")!=1){

imgName=fileName;

}

}

//获取到资源文件名称以后

InputStream resourceAsStream = processEngine.getRepositoryService().getResourceAsStream(deploymentId, imgName);

FileOutputStream outputStream=**new** FileOutputStream("d://"+imgName);

**byte**[] buf=**new** **byte**[1024];

**int** len=0;

**while**((len=resourceAsStream.read(buf))!=-1){

outputStream.write(buf, 0, len);

outputStream.flush();

}

}

#### 5.1.6 启动流程实例

**public** **void** start(){

RuntimeService runtimeService = buildProcessEngine.getRuntimeService();

ProcessInstance startProcessInstanceByKey = runtimeService.startProcessInstanceByKey("myProcess");

System.***out***.println("流程实例id"+startProcessInstanceByKey.getId());

System.***out***.println("流程定义id"+startProcessInstanceByKey.getProcessDefinitionId());

}

SELECT \* FROM act\_ru\_execution; //流程定义表

SELECT \* FROM act\_ru\_task;//任务表

#### 5.1.7 删除流程定义

**public** **void** deleteProcess(){

String deployId="1";

processEngine.getRepositoryService().deleteDeployment(deployId);

}

#### 5.1.8 查询历史流程(已结束未结束)

@Test

**public** **void** findHistory(){

HistoryService historyService = processEngine.getHistoryService();

HistoricProcessInstanceQuery createHistoricProcessInstanceQuery = historyService.createHistoricProcessInstanceQuery();

List<HistoricProcessInstance> list = createHistoricProcessInstanceQuery.list();

**if**(list.size()>0){

**for**(HistoricProcessInstance historicProcessInstance:list){

System.***out***.println("历史流程id:"+historicProcessInstance.getId());

System.***out***.println("历史流程实例id:"+historicProcessInstance.getProcessDefinitionId());

System.***out***.println("历史流程起始时间:"+historicProcessInstance.getStartTime());

System.***out***.println("历史流程结束时间:"+historicProcessInstance.getEndTime());

}

}

}

SELECT \* FROM act\_hi\_procinst;

### 5.2 任务

#### 5.2.1 任务查询

**public** **void** findTask(){

TaskService taskService = buildProcessEngine.getTaskService();

TaskQuery createTaskQuery = taskService.createTaskQuery();

createTaskQuery.taskAssignee("张三");

List<Task> list = createTaskQuery.list();

**for**(Task t:list){

System.***out***.println("任务id"+t.getId());

System.***out***.println("任务名称"+t.getName());

System.***out***.println("任务受理人"+t.getAssignee());

}

}

SELECT \* FROM act\_ru\_task;//任务表

#### 5.2.2 办理任务

**public** **void** competeTask(){

String taskid="2504";

TaskService taskService = buildProcessEngine.getTaskService();

taskService.complete(taskid);

System.***out***.println("完成任务");

}

SELECT \* FROM act\_ru\_task;//任务表 任务办理完以后会清除任务表

Select \* from act\_ru\_execution //清除流程实例表

#### 5.2.3 指定流程活动的任务

@Test

**public** **void** findRunTask(){

String processInstanceId="5001";

RuntimeService runtimeService = processEngine.getRuntimeService();

ProcessInstanceQuery createProcessInstanceQuery = runtimeService.createProcessInstanceQuery();

ProcessInstanceQuery processInstanceId2 = createProcessInstanceQuery.processInstanceId(processInstanceId);

ProcessInstance singleResult = processInstanceId2.singleResult();

String activityId = singleResult.getActivityId();

System.***out***.println("获取指定流程实例对应正在活动的任务 流程实例:"+processInstanceId+"任务:"+activityId);

}

Select \* from act\_ru\_execution //清除流程实例表

#### 5.2.4 查询历史任务

**public** **void** findHistoryTask(){

String processInstanceId="5001";

HistoricTaskInstanceQuery createHistoricTaskInstanceQuery = processEngine.getHistoryService().createHistoricTaskInstanceQuery();

createHistoricTaskInstanceQuery.processInstanceId(processInstanceId);

List<HistoricTaskInstance> list = createHistoricTaskInstanceQuery.list();

**if**(list.size()>0){

**for**(HistoricTaskInstance li:list){

System.***out***.println("历史任务id:"+li.getId());

System.***out***.println("历史流程实例id:"+li.getProcessInstanceId());

System.***out***.println("历史流程起始时间:"+li.getStartTime());

System.***out***.println("历史流程结束时间:"+li.getEndTime());

}

}

}

SELECT \* FROM act\_hi\_taskinst;

### 5.3 流程变量（对象二进制存）

#### 5.3.1 变量作用

流程变量主要用来解决的问题

1 携带流程数据

2 动态绑定流程节点分配目标

3 推动分之流程的创建

4 流程变量如果是对象类型，要实现序列化接口，并且他是一二进制流的方式保存在二进制表中的

#### 5.3.2 设置变量

**public** **void** setFlowVar(){

String exxcutionId="xxxId";//流程实例id

String processDefId="processdefId";//流程定义id

String taskId="taskId";//任务id

String varName="varname";

String varValue="varvalue";

Map varValues=**new** HashMap();

//为流程对象设置 流程变量

RuntimeService runtimeService = processEngine.getRuntimeService();

runtimeService.setVariable(exxcutionId, varName, varValue);

runtimeService.setVariables(exxcutionId, varValues);

runtimeService.setVariableLocal(exxcutionId, varName, varValue);

runtimeService.setVariablesLocal(exxcutionId,varValues );

//在流程启动的时候设置流程变量

runtimeService.startProcessInstanceById(processDefId, varValues);

//为任务设置流程变量

TaskService taskService = processEngine.getTaskService();

taskService.setVariable(taskId, varName, varValue);

taskService.setVariables(taskId, varValues);

taskService.setVariableLocal(taskId, varName, varValue);

taskService.setVariables(taskId, varValues);

}

#### 5.3.3 获取变量

**public** **void** getVar(){

String exxcutionId="xxxId";//流程实例id

String processDefId="processdefId";//流程定义id

String taskId="taskId";//任务id

String varName="varname";

//获取执行对象中的流程变量

RuntimeService runtimeService = processEngine.getRuntimeService();

Object variable = runtimeService.getVariable(exxcutionId, varName);//获得流程实例变量

Map<String, Object> variables = runtimeService.getVariables(varName);//获取流程实例变量

Object variableLocal = runtimeService.getVariableLocal(exxcutionId, varName);//获取流程任务变量

Map<String, Object> variablesLocal = runtimeService.getVariablesLocal(varName);//获取流程任务变量

//从流程实例中获取变量

TaskService taskService = processEngine.getTaskService();

Object variable2 = taskService.getVariable(taskId, varName);

Map<String, Object> variables2 = taskService.getVariables(varName);

//从流程任务中获取变量

Object variableLocal2 = taskService.getVariableLocal(taskId, varName);

Map<String, Object> variablesLocal2 = taskService.getVariablesLocal(varName);

}

SELECT \* FROM act\_ru\_variable;

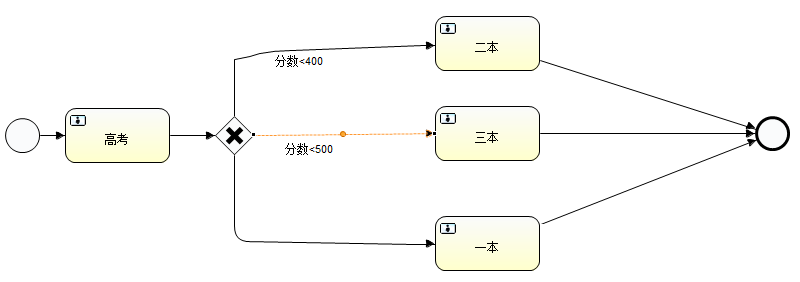
SELECT \* FROM act\_hi\_varinst;

SELECT \* FROM act\_ge\_bytearray;

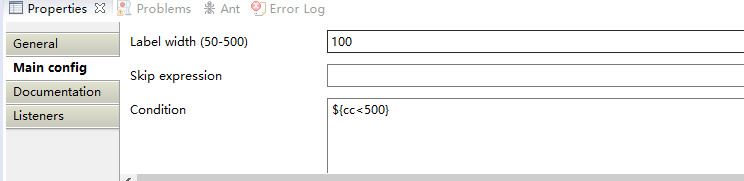
### 5.4 排它网关（分之结构）

#### 5.4.1 绘制流程

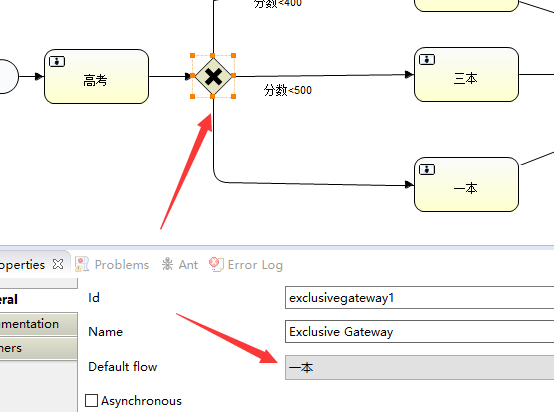
解决选择性的问题建模



#### 5.4.2 设置条件变量

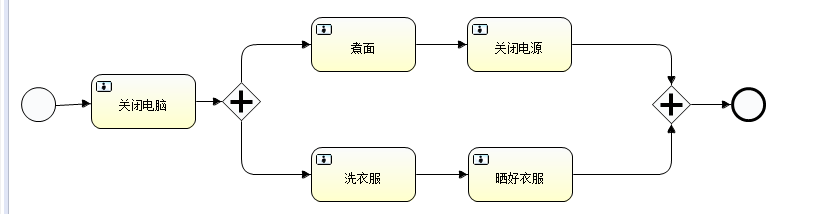


#### 5.4.3 默认分支



#### 5.4.4 并行网关

解决并行性得的问题建模



### 5.5 创建组

**public** **void** testCGroup(){

IdentityService identityService = processengine.getIdentityService();

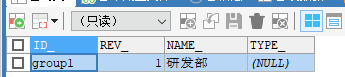
Group group=**new** GroupEntity();

group.setId("group1");

group.setName("研发部");

identityService.saveGroup(group);

}



### 5.6 创建用户

@Test

**public** **void** testCUser(){

IdentityService identityService = processengine.getIdentityService();

User user=**new** UserEntity();

User user1=**new** UserEntity();

user1.setId("u1");

user1.setFirstName("user1");

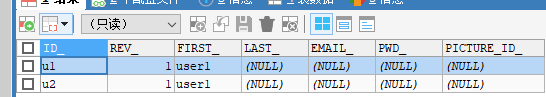
user.setId("u2");

user.setFirstName("user1");

identityService.saveUser(user);

identityService.saveUser(user1);

}



### 5.7 组与用户关系

@Test

**public** **void** testCMem(){

IdentityService identityService = processengine.getIdentityService();

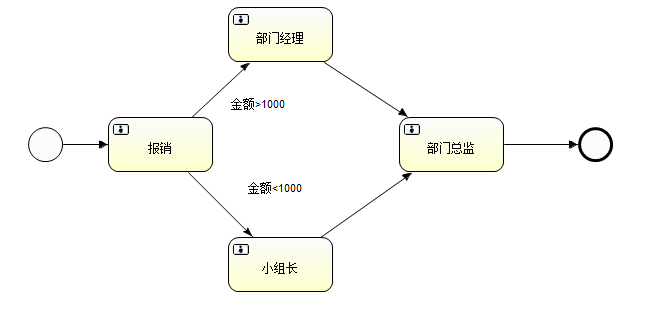
identityService.createMembership("u2","group1");

}

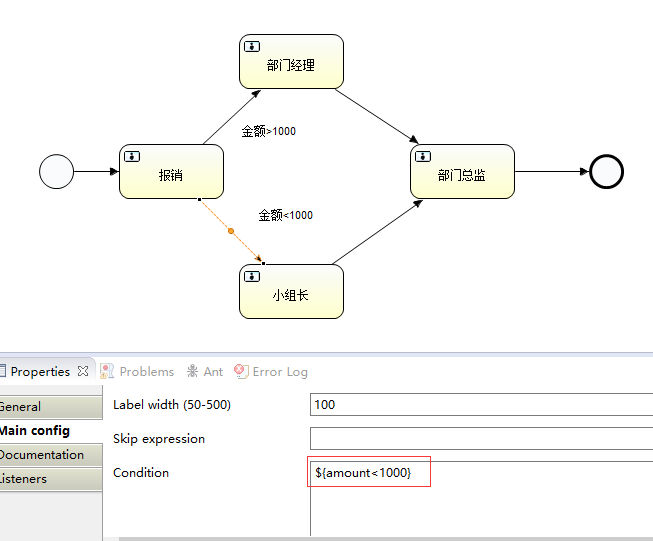
## 六 案例

### 6.1 案例一 基本选择流程

#### 6.1.1 绘制流程图



#### 6.1.2 设置流程变量



如上所示${}参数符 里面放一个boolean变量

${a=b}

#{a=’b’}

#### 6.1.3 加载并部署流程

@Before

**public** **void** init(){

ProcessEngineConfiguration createProcessEngineConfigurationFromResource =

ProcessEngineConfiguration.

*createProcessEngineConfigurationFromResource*("activiti.cfg.xml");

processengine = createProcessEngineConfigurationFromResource

.buildProcessEngine();

}

@Test

**public** **void** deploy(){

RepositoryService repositoryService = processengine.getRepositoryService();

DeploymentBuilder createDeployment = repositoryService.createDeployment();

InputStream inputStream=getClass().getClassLoader().

getResourceAsStream("amount.zip");

ZipInputStream zipInputStream=**new** ZipInputStream(inputStream);

createDeployment.addZipInputStream(zipInputStream);

createDeployment.deploy();

}

#### 6.1.4 启动流程

**public** **void** start(){

String processDefkey="amountPro";

RuntimeService runtimeService = processengine.getRuntimeService();

runtimeService.startProcessInstanceByKey(processDefkey);

}

#### 6.1.5 报销办理

@Test

**public** **void** completeT(){

String taskId="32504";

TaskService taskService = processengine.getTaskService();

taskService.setVariable(taskId, "amount", 1200);

taskService.complete(taskId);

}

#### 6.1.6 查询任务

@Test

**public** **void** findTaskList(){

String executionId="32501";

TaskService taskService = processengine.getTaskService();

TaskQuery createTaskQuery = taskService.createTaskQuery();

List<Task> list = createTaskQuery.executionId(executionId).list();

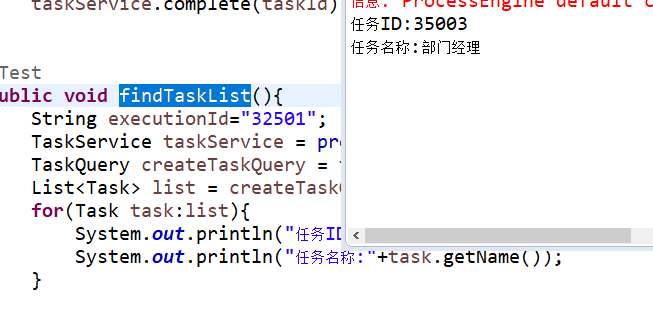
**for**(Task task:list){

System.***out***.println("任务ID:"+task.getId());

System.***out***.println("任务名称:"+task.getName());

}

}



#### 6.1.7 部门经理办理

@Test

**public** **void** completeT(){

String taskId="35003";

TaskService taskService = processengine.getTaskService();

taskService.setVariable(taskId, "amount", 1200);

taskService.complete(taskId);

}

#### 6.1.8 查询任务

**public** **void** findTaskList(){

String executionId="32501";

TaskService taskService = processengine.getTaskService();

TaskQuery createTaskQuery = taskService.createTaskQuery();

List<Task> list = createTaskQuery.executionId(executionId).list();

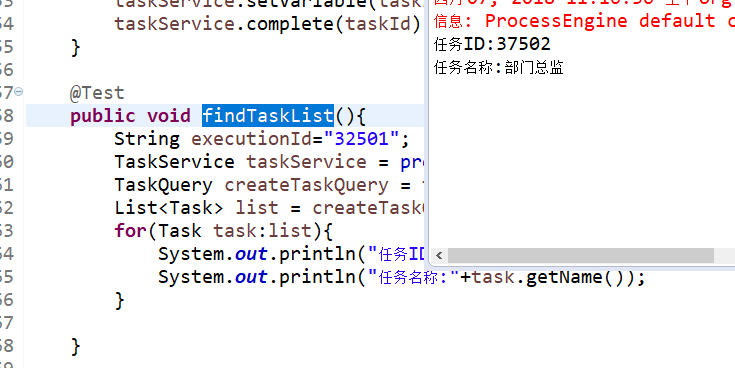
**for**(Task task:list){

System.***out***.println("任务ID:"+task.getId());

System.***out***.println("任务名称:"+task.getName());

}

}



#### 6.1.9 部门总监办理

@Test

**public** **void** completeT(){

String taskId="37502";

TaskService taskService = processengine.getTaskService();

taskService.setVariable(taskId, "amount", 1200);

taskService.complete(taskId);

}

#### 6.1.10 查询历史任务

@Test

**public** **void** findHistoryTask(){

String hisexecuid="32501";

HistoricTaskInstanceQuery createHistoricTaskInstanceQuery = processengine.getHistoryService().createHistoricTaskInstanceQuery();

List<HistoricTaskInstance> list = createHistoricTaskInstanceQuery.executionId(hisexecuid).list();

**for**(HistoricTaskInstance instance:list){

System.***out***.println("任务id:"+instance.getId()+" 流程名称:"+instance.getName());

System.***out***.println("开始时间:"+instance.getStartTime()+" 结束时间:"+instance.getEndTime());

}

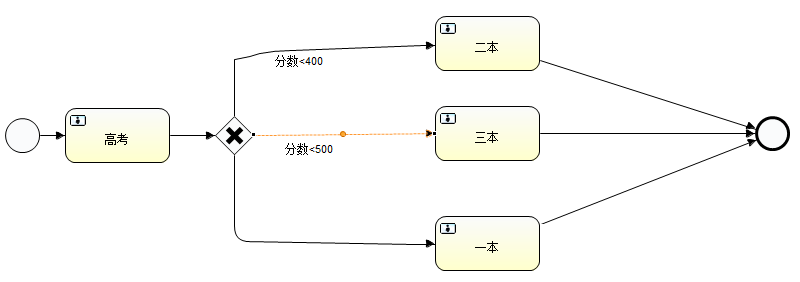
}



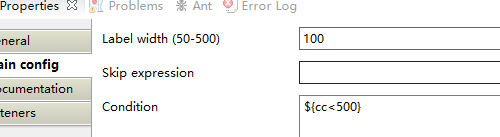
#### 6.1.11 结束流程

### 6.2 案例二 排他网关流程

#### 6.2.1 绘制流程图



#### 6.2.2 设置流程条件变量



#### 6.2.3 初始部署流程

@Before

**public** **void** init(){

ProcessEngineConfiguration createProcessEngineConfigurationFromResource =

ProcessEngineConfiguration.

*createProcessEngineConfigurationFromResource*("activiti.cfg.xml");

processengine = createProcessEngineConfigurationFromResource

.buildProcessEngine();

}

@Test

**public** **void** deploy(){

RepositoryService repositoryService = processengine.getRepositoryService();

DeploymentBuilder createDeployment = repositoryService.createDeployment();

InputStream inputStream=getClass().getClassLoader().

getResourceAsStream("exclusiveGate.zip");

ZipInputStream zipInputStream=**new** ZipInputStream(inputStream);

createDeployment.addZipInputStream(zipInputStream);

createDeployment.deploy();

}

#### 6.2.4 启动流程

@Test

**public** **void** start(){

String processDefkey="高考择校";

RuntimeService runtimeService = processengine.getRuntimeService();

runtimeService.startProcessInstanceByKey(processDefkey);

}

#### 6.2.5 执行流程(默认网关流程变量)

@Test

**public** **void** completeT(){

String taskId="45004";

TaskService taskService = processengine.getTaskService();

taskService.setVariable(taskId, "cc", 550);

taskService.complete(taskId);

}

#### 6.2.6 查询任务

@Test

**public** **void** findTaskList(){

String executionId="45001";

TaskService taskService = processengine.getTaskService();

TaskQuery createTaskQuery = taskService.createTaskQuery();

List<Task> list = createTaskQuery.executionId(executionId).list();

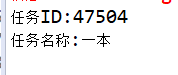
**for**(Task task:list){

System.***out***.println("任务ID:"+task.getId());

System.***out***.println("任务名称:"+task.getName());

}

}



#### 6.2.7 执行任务

@Test

**public** **void** completeT(){

String taskId="47504";

TaskService taskService = processengine.getTaskService();

taskService.setVariable(taskId, "cc", 550);

taskService.complete(taskId);

}

#### 6.2.8 结束并查询历史任务

@Test

**public** **void** findHistoryTask(){

String hisexecuid="45001";

HistoricTaskInstanceQuery createHistoricTaskInstanceQuery = processengine.getHistoryService().createHistoricTaskInstanceQuery();

List<HistoricTaskInstance> list = createHistoricTaskInstanceQuery.executionId(hisexecuid).list();

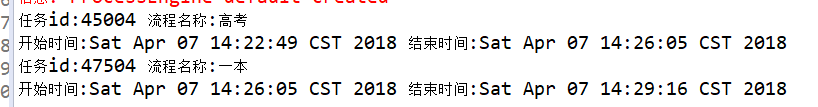
**for**(HistoricTaskInstance instance:list){

System.***out***.println("任务id:"+instance.getId()+" 流程名称:"+instance.getName());

System.***out***.println("开始时间:"+instance.getStartTime()+" 结束时间:"+instance.getEndTime());

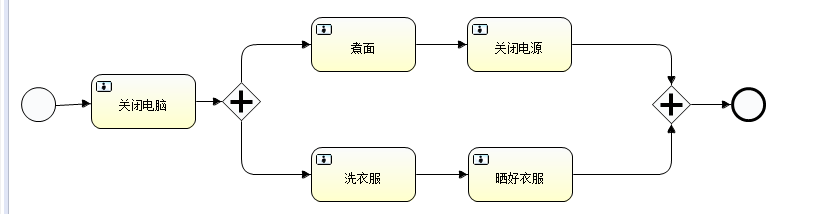
}

}



### 6.3 案例三 并行网关流程

6.3.1 绘制流程图



6.3.2 初始化与部署

ProcessEngine processengine=**null**;

@Before

**public** **void** init(){

ProcessEngineConfiguration createProcessEngineConfigurationFromResource =

ProcessEngineConfiguration.

*createProcessEngineConfigurationFromResource*("activiti.cfg.xml");

processengine = createProcessEngineConfigurationFromResource

.buildProcessEngine();

}

@Test

**public** **void** deploy(){

RepositoryService repositoryService = processengine.getRepositoryService();

DeploymentBuilder createDeployment = repositoryService.createDeployment();

InputStream inputStream=getClass().getClassLoader().

getResourceAsStream("paralle.zip");

ZipInputStream zipInputStream=**new** ZipInputStream(inputStream);

createDeployment.addZipInputStream(zipInputStream);

createDeployment.deploy();

}

6.3.3 启动流程

@Test

**public** **void** start(){

String processDefkey="myProcess";

RuntimeService runtimeService = processengine.getRuntimeService();

runtimeService.startProcessInstanceByKey(processDefkey);

}

6.3.4 关闭电脑（两个执行对象）

@Test

**public** **void** completeT(){

String taskId="62504";

TaskService taskService = processengine.getTaskService();

taskService.setVariable(taskId, "cc", 550);

taskService.complete(taskId);

}



关闭电后，出现了两个执行对象，办理了煮面的任务之后，数据库中剩下流程可见，同一个流程实例52501，参上了两个执行对

象65003,65004 晒衣服执行完以后流程对象结束了，但是流程实例还没有结束，还再跑，直到上面的关闭电源，结束以后

整个过程才真正的结束了。

6.3.5 查询任务

@Test

**public** **void** findHistoryTask(){

String instanceId="77501";

HistoricTaskInstanceQuery createHistoricTaskInstanceQuery = processengine.getHistoryService().createHistoricTaskInstanceQuery();

List<HistoricTaskInstance> list = createHistoricTaskInstanceQuery.processInstanceId(instanceId).list();

**for**(HistoricTaskInstance instance:list){

System.***out***.println("任务id:"+instance.getId()+" 流程名称:"+instance.getName());

System.***out***.println("开始时间:"+instance.getStartTime()+" 结束时间:"+instance.getEndTime());

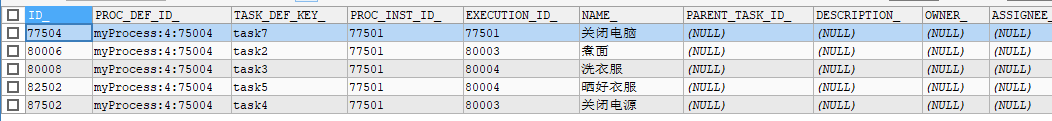
}

}

通过流程实例id查询对应的任务



数据库中表结构

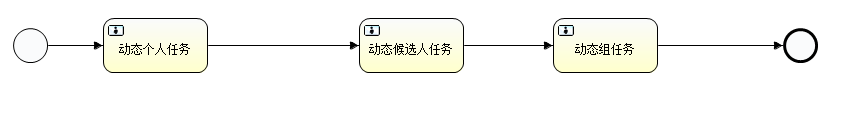


如上所示从表中可以看出一个流程实例，产生了三个执行对象，与5个任务，产生了两条路径

分别是77501,80003;7501,80004

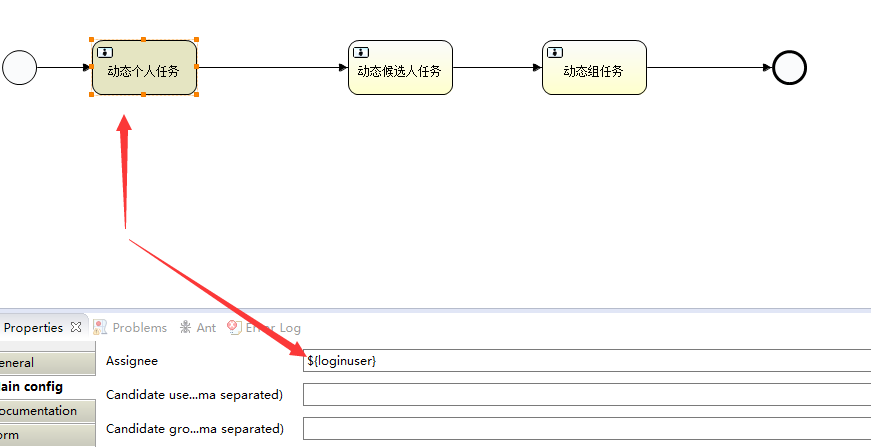
### 6.4 案例四 个人 候选人 组

#### 6.4.1 绘制流程图

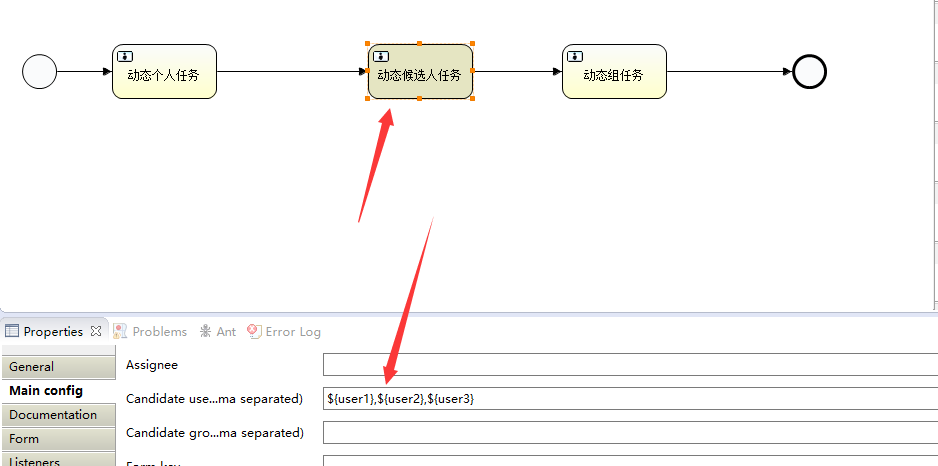


#### 6.4.2 设置流程变量

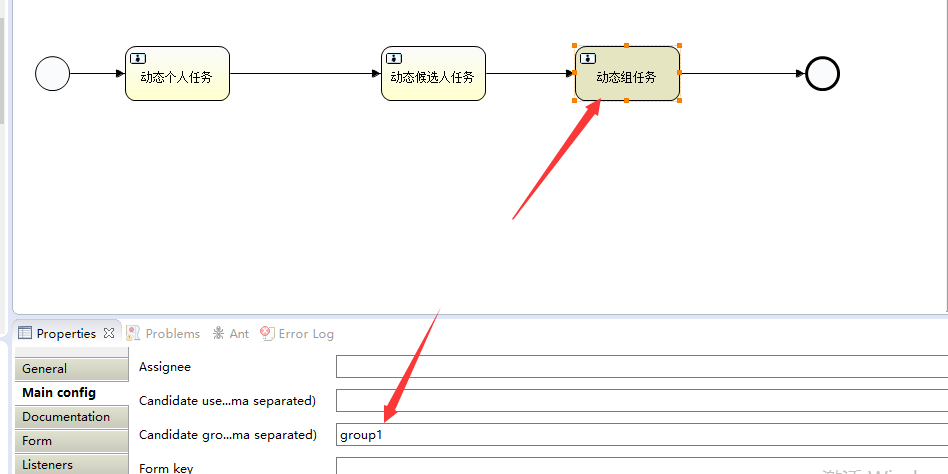
个人任务



候选人任务



组任务



#### 6.4.3 部署与初始化

@Before

**public** **void** init(){

ProcessEngineConfiguration createProcessEngineConfigurationFromResource =

ProcessEngineConfiguration.

*createProcessEngineConfigurationFromResource*("activiti.cfg.xml");

processengine = createProcessEngineConfigurationFromResource

.buildProcessEngine();

}

@Test

**public** **void** deploy(){

RepositoryService repositoryService = processengine.getRepositoryService();

DeploymentBuilder createDeployment = repositoryService.createDeployment();

createDeployment.addClasspathResource("diagrams/person.bpmn")

.addClasspathResource("diagrams/person.png").deploy();

}

#### 6.4.4 启动流程

@Test

**public** **void** start(){

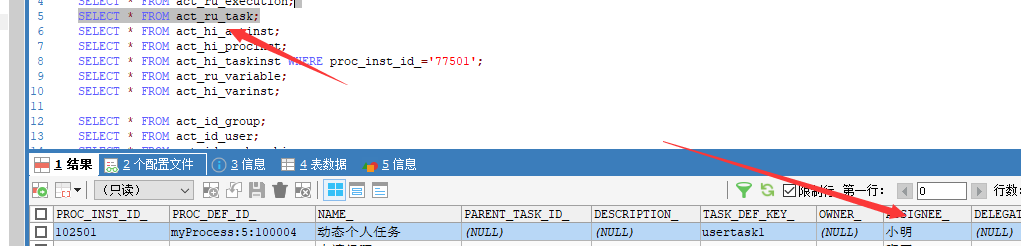
Map param=**new** HashMap();

param.put("loginuser", "小明");

RuntimeService runtimeService = processengine.getRuntimeService();

runtimeService.startProcessInstanceByKey("myProcess",param);

}



如上任务表中的指定人已经动态做了改变

#### 6.4.5 查询并办理 指定候选人

@Test

**public** **void** queryTask(){

TaskService taskService = processengine.getTaskService();

TaskQuery createTaskQuery = taskService.createTaskQuery();

TaskQuery taskAssignee = createTaskQuery.taskAssignee("小明");

List<Task> list = taskAssignee.list();

**for**(Task task:list){

System.***out***.println("任务id:"+task.getId());

System.***out***.println("任务id:"+task.getName());

}

}

@Test

**public** **void** comlepxTask(){

TaskService taskService = processengine.getTaskService();

Map users=**new** HashMap();

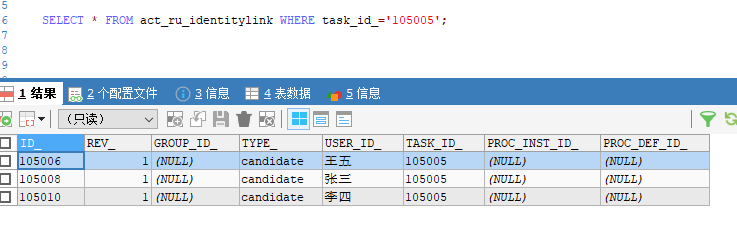
users.put("user1", "张三");

users.put("user2", "李四");

users.put("user3", "王五");

taskService.complete("102505",users);

}



涉及的表act\_ru+identitylink

#### 6.4.6 候选人拾取任务

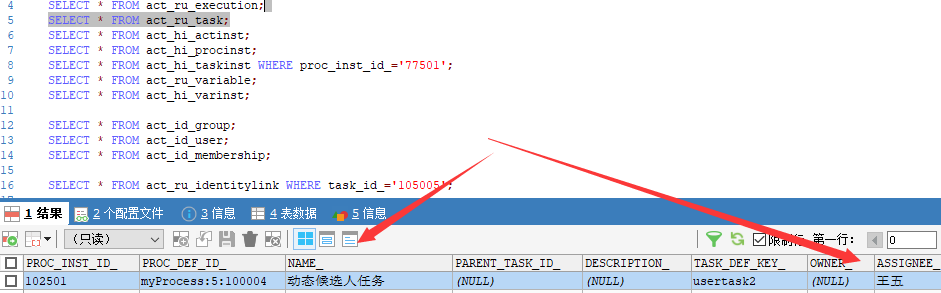
@Test

**public** **void** claimTask(){

TaskService taskService = processengine.getTaskService();

taskService.claim("105005", "王五");

}



此时只要是候选人都可以拾取相应的任务

#### 6.4.7 候选人办理

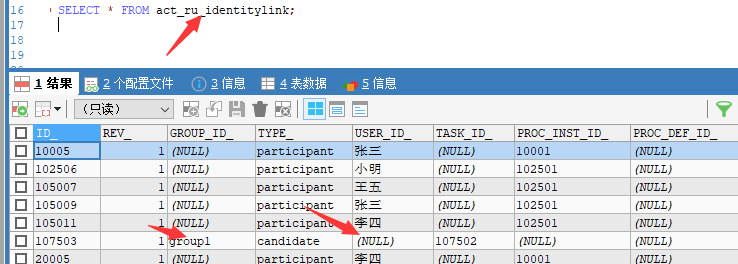
@Test

**public** **void** complexTax(){

TaskService taskService = processengine.getTaskService();

taskService.complete("105005");

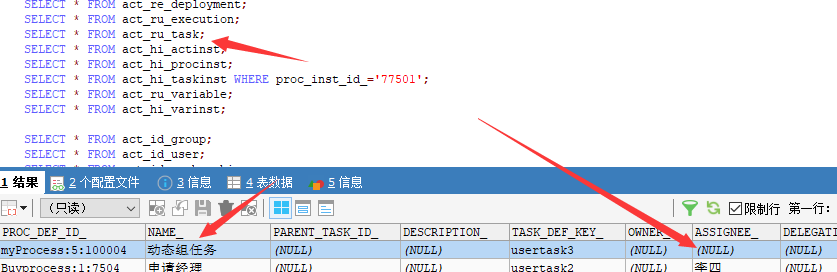
}



此时候选人中没有值了，只有获选组了。

#### 6.4.8 同组用户拾取并办理任务

拾取前



拾取后



## 七 spring集成activiti

### 7.1 jar包依赖

<!-- activiti -->

<dependency>

<groupId>org.activiti</groupId>

<artifactId>activiti-engine</artifactId>

<version>${activiti}</version>

</dependency>

<!-- activiti 与 Spring 集成 -->

<dependency>

<groupId>org.activiti</groupId>

<artifactId>activiti-spring</artifactId>

<version>${activiti}</version>

</dependency>

<activiti>5.15.1</activiti>

### 7.2 spring配置

<!-- 事务管理器 -->

<bean id=*"transactionManager"*

class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>

<property name=*"dataSource"* ref=*"dataSource"* />

</bean>

<!-- 通知 -->

<tx:advice id=*"txAdvice"* transaction-manager=*"transactionManager"*>

<tx:attributes>

<tx:method name=*"save\*"* propagation=*"REQUIRED"* rollback-for=*"java.lang.Exception"*/>

<tx:method name=*"insert\*"* propagation=*"REQUIRED"* rollback-for=*"java.lang.Exception"*/>

<tx:method name=*"add\*"* propagation=*"REQUIRED"* rollback-for=*"java.lang.Exception"*/>

<tx:method name=*"create\*"* propagation=*"REQUIRED"* rollback-for=*"java.lang.Exception"*/>

<tx:method name=*"delete\*"* propagation=*"REQUIRED"* rollback-for=*"java.lang.Exception"*/>

<tx:method name=*"update\*"* propagation=*"REQUIRED"* rollback-for=*"java.lang.Exception"*/>

<tx:method name=*"find\*"* propagation=*"SUPPORTS"* read-only=*"true"* />

<tx:method name=*"select\*"* propagation=*"SUPPORTS"* read-only=*"true"* />

<tx:method name=*"get\*"* propagation=*"SUPPORTS"* read-only=*"true"* />

</tx:attributes>

</tx:advice>

<!-- 切面 -->

<aop:config>

<aop:advisor advice-ref=*"txAdvice"* pointcut=*"execution(\* yonyou.worn.service.\*.\*(..))"* />

</aop:config>

<!-- 邮件工具 -->

<bean id=*"mailSenderImpl"* class=*"org.springframework.mail.javamail.JavaMailSenderImpl"*></bean>

<bean id=*"simpleMailMessage"* class=*"org.springframework.mail.SimpleMailMessage"*></bean>

<!-- 配置事务 -->

<bean id=*"txManager"*

class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>

<property name=*"dataSource"* ref=*"dataSource"*></property>

</bean>

<!--使用基于注解方式配置事务 -->

<tx:annotation-driven transaction-manager=*"txManager"* />

<!-- 加载activiti引擎 -->

<bean id=*"processEngine"* class=*"org.activiti.spring.ProcessEngineFactoryBean"*>

<property name=*"processEngineConfiguration"* ref=*"processEngineConfiguration"* />

</bean>

<bean id=*"processEngineConfiguration"* class=*"org.activiti.spring.SpringProcessEngineConfiguration"*>

<property name=*"dataSource"* ref=*"dataSource"* />

<property name=*"transactionManager"* ref=*"transactionManager"* />

<property name=*"databaseSchemaUpdate"* value=*"true"* />

<property name=*"jobExecutorActivate"* value=*"false"* />

</bean>

<!-- activiti的各种服务接口 -->

<bean id=*"repositoryService"* factory-bean=*"processEngine"*

factory-method=*"getRepositoryService"* />

<bean id=*"runtimeService"* factory-bean=*"processEngine"*

factory-method=*"getRuntimeService"* />

<bean id=*"taskService"* factory-bean=*"processEngine"*

factory-method=*"getTaskService"* />

<bean id=*"historyService"* factory-bean=*"processEngine"*

factory-method=*"getHistoryService"* />

<bean id=*"managementService"* factory-bean=*"processEngine"*

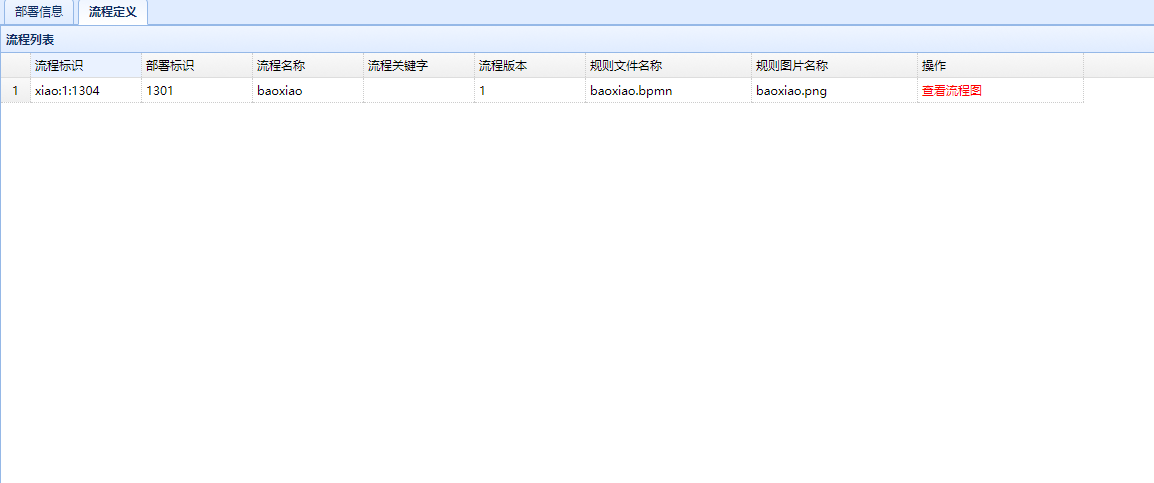
factory-method=*"getManagementService"* />

### 7.3 原型设计

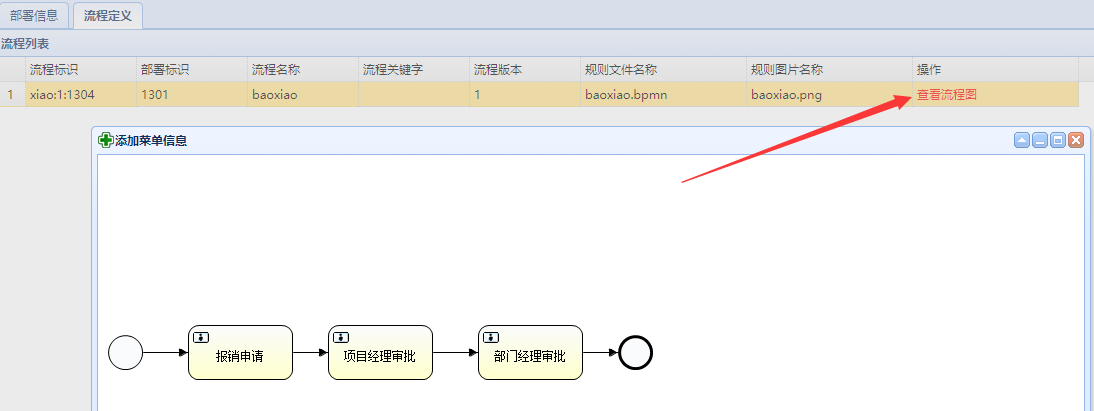
#### 7.3.1 流程的部署



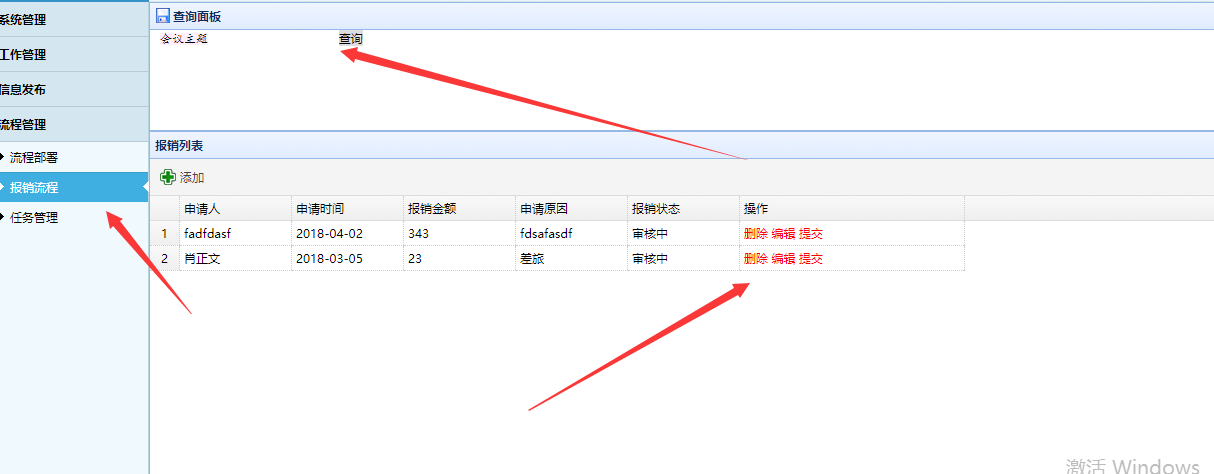
#### 7.3.2 流程的定义



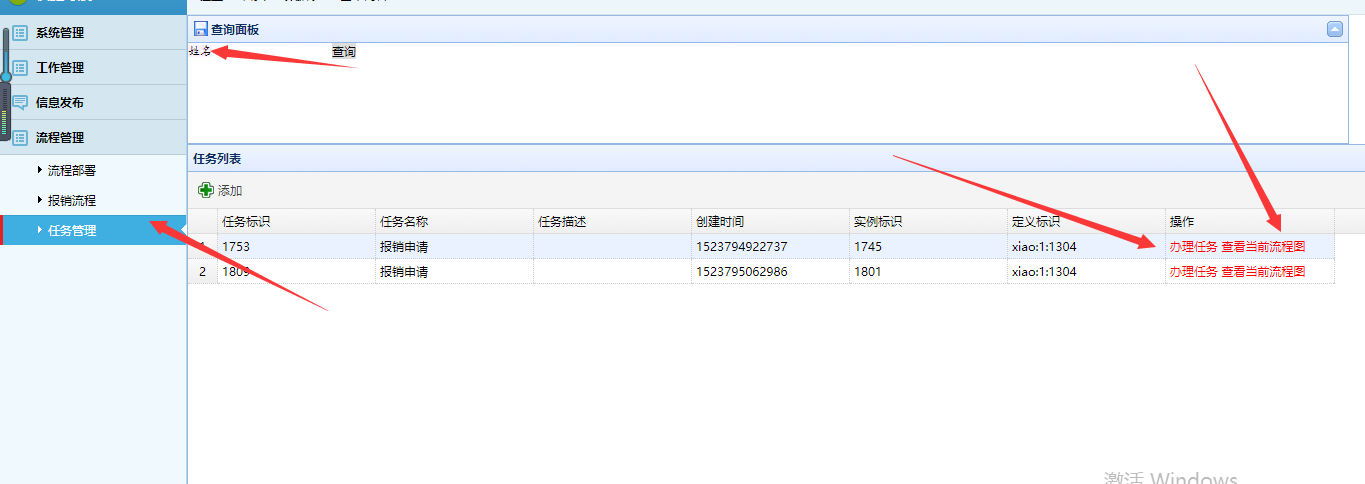
#### 7.3.3 流程定义图



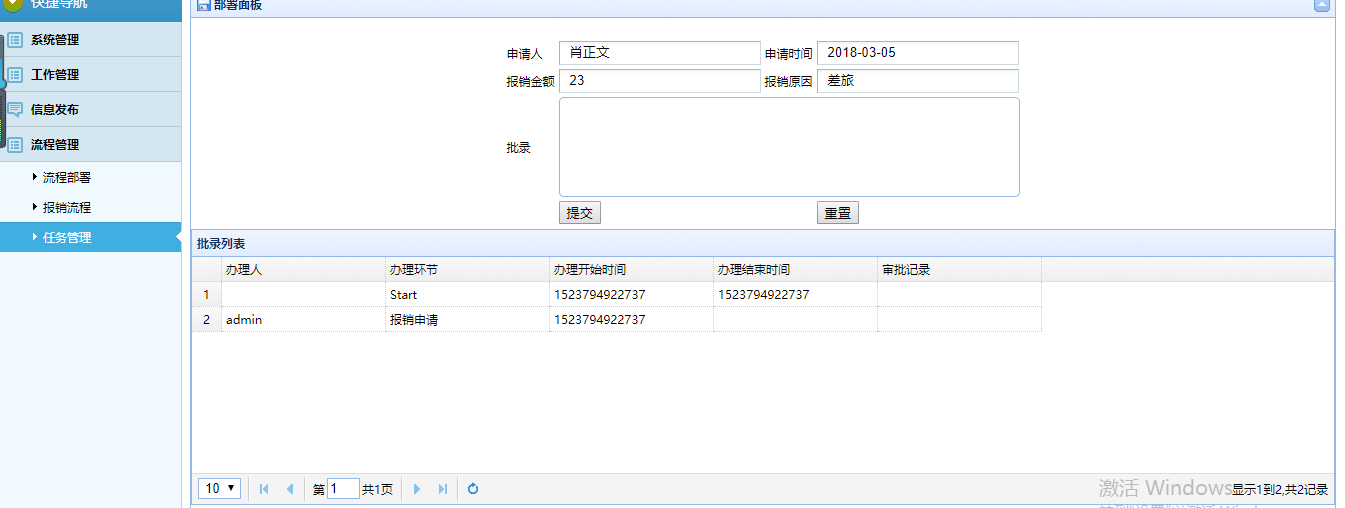
#### 7.3.4 业务流程



#### 7.3.5 任务管理



#### 7.3.6 任务办理



#### 7.3.7 查看当前任务流程图

