# 尚马教育 JAVA 高级课程

# Activiti工作流引擎

文档编号：C15

创建日期： 2021-02-26

最后修改日期：2021-02-26

版 本 号：V1.0

电子版文件名：尚马教育-第三阶段-15.activiti工作流引擎.docx

**文档修改记录：**

|  |  |  |  |
| --- | --- | --- | --- |
| 更新日期 | 更新作者 | 更新说明 | 版本号 |
| 2021-02-26 | 冯勇涛 |  | V1.0 |

目录

[尚马教育 JAVA 高级课程 1](#_Toc4958)

[Activiti工作流引擎 1](#_Toc26886)

[1. Activiti介绍 3](#_Toc9880)

[1.1. 认识activiti 3](#_Toc27767)

[1.2. 认识BPMN 3](#_Toc12424)

[2. 环境安装 3](#_Toc12024)

[2.1. 数据库环境 4](#_Toc28849)

[2.2. 流程设计之Eclipse插件 7](#_Toc23963)

[2.3. 代码运行环境 11](#_Toc1290)

[3. IdentityService服务 13](#_Toc30438)

[4. 流程设计 15](#_Toc4295)

[5. RepositoryService服务 17](#_Toc15608)

[6. 创建请假申请单业务表与实体类 19](#_Toc12892)

[7. RuntimeService服务 20](#_Toc28155)

[8. TaskService服务 21](#_Toc2729)

[8.1. 查询待办任务 21](#_Toc7468)

[8.2. 多人任务签收 22](#_Toc26223)

[8.3. 签收后审批 22](#_Toc6384)

[9. HistoryService流程跟踪 23](#_Toc25725)

[9.1. 申请人查询历史任务 24](#_Toc28375)

[9.2. 审批人查询已办任务 25](#_Toc1590)

## Activiti介绍

### 认识activiti

Activiti基于Apache许可的开源BPM（业务流程管理）平台，创始人Tom Baeyens是JBoss jBPM的项目架构师，因此activiti很多思想与jbpm流程管理框架类似，activiti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。

在企业日常运营中，企业业务流程审批是非常常见的，比如：财务报销流程，贴发票到报销单，提交给组长，组长审批后提交主管，主管审批后提交财务部，财务部主管审批后，向申请人银行卡打钱。这样一个财务报销流程就执行完成。

### 认识BPMN

Bpmn是业务流程建模与标注（Business Process Model and Notation，BPMN) 的一种标记语言，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），activiti的eclipse插件即是一套bpmn流程定义的实现插件。

## 环境安装

Ssm环境下，Activiti目前主流应用版本5.22。

版本7主要针对springboot提供支持。

下载地址：

<https://github.com/Activiti/Activiti/releases/download/activiti-5.22.0/activiti-5.22.0.zip>

解压压缩包后，如下图：

|  |
| --- |
|  |

### 数据库环境

找到database文件夹下的created脚本，执行mysql数据库的三个脚本文件。

|  |
| --- |
|  |

创建成功后，数据库有25张表：

|  |
| --- |
|  |

数据库表介绍：

表名默认以“ACT\_”开头,并且表名的第二部分用两个字母表明表的用例，而这个用例也基本上跟Service API匹配。

ACT\_GE\_\* : “GE”代表“General”（通用），用在各种情况下；

ACT\_HI\_\* : “HI”代表“History”（历史），这些表中保存的都是历史数据，比如执行过的流程实例、变量、任务，等等。Activit默认提供了4种历史级别：

ACT\_ID\_\* : “ID”代表“Identity”（身份），这些表中保存的都是身份信息，如用户和组以及两者之间的关系。如果Activiti被集成在某一系统当中的话，这些表可以不用，可以直接使用现有系统中的用户或组信息；

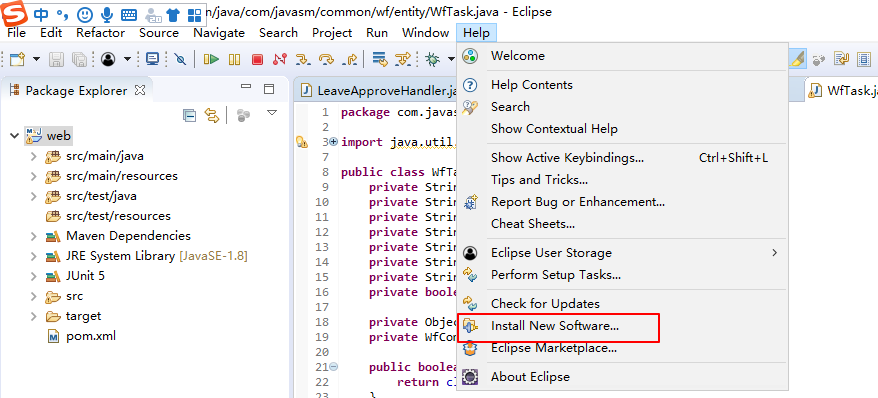
ACT\_RE\_\* : “RE”代表“Repository”（仓库），这些表中保存一些‘静态’信息，如流程定义和流程资源（如图片、规则等）；

ACT\_RU\_\* : “RU”代表“Runtime”（运行时），这些表中保存一些流程实例、用户任务、变量等的运行时数据。Activiti只保存流程实例在执行过程中的运行时数据，并且当流程结束后会立即移除这些数据，这是为了保证运行时表尽量的小并运行的足够快；

| **表分类** | **表名称** | **表含义** |
| --- | --- | --- |
|  | act\_evt\_log | 事件处理日志表 |
| 一般数据 | act\_ge\_bytearray | 通用的流程定义和流程资源 |
|  | act\_ge\_property | 系统相关属性 |
| 流程历史记录 | act\_hi\_actinst | 历史的流程实例 |
|  | act\_hi\_attachment | 历史的流程附件 |
|  | act\_hi\_comment | 历史的说明性信息 |
|  | act\_hi\_detail | 历史的流程运行中的细节信息 |
|  | act\_hi\_identitylink | 历史的流程运行过程中用户关系 |
|  | act\_hi\_procinst | 历史的流程实例 |
|  | act\_hi\_taskinst | 历史的任务实例 |
|  | act\_hi\_varinst | 历史的流程运行中的变量信息 |
| 用户用户组表 | act\_id\_group | 身份信息-组信息 |
|  | act\_id\_info | 身份信息-组信息 |
|  | act\_id\_membership | 身份信息-用户和组关系的中间表 |
|  | act\_id\_user | 身份信息-用户信息 |
|  | act\_procdef\_info | 死信任务 |
| 流程定义表 | act\_re\_deployment | 部署单元信息 |
|  | act\_re\_model | 模型信息 |
|  | act\_re\_procdef | 已部署的流程定义 |
| 运行实例表 | act\_ru\_deadletter\_job | 执行失败任务表 |
|  | act\_ru\_event\_subscr | 运行时事件 |
|  | act\_ru\_execution | 运行时流程执行实例 |
|  | act\_ru\_identitylink | 运行时用户关系信息 |
|  | act\_ru\_job | 运行时作业 |
|  | act\_ru\_suspended\_job | 运行时暂停任务 |
|  | act\_ru\_task | 运行时任务 |
|  | act\_ru\_timer\_job | 运行时定时任务 |
|  | act\_ru\_variable | 运行时变量表 |

### 流程设计之Eclipse插件

第一步：点击eclipse上方工具栏的Help，选择Install New Software。

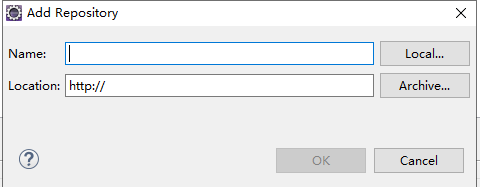


第二步：在弹出窗口下点击add按钮，填写插件名称与安装地址。

* 在线安装：

Name：Activiti BPMN 2.O designer

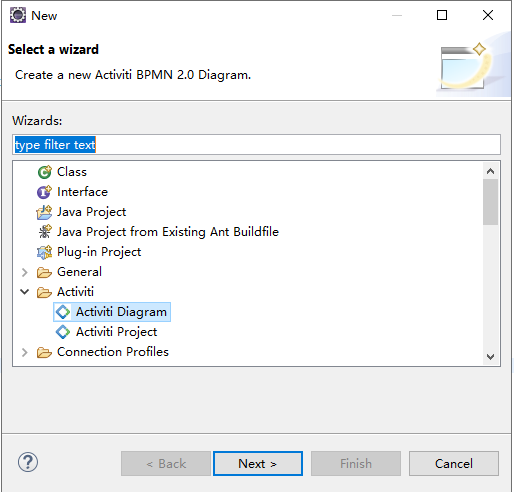
Location：<http://activiti.org/designer/update/>

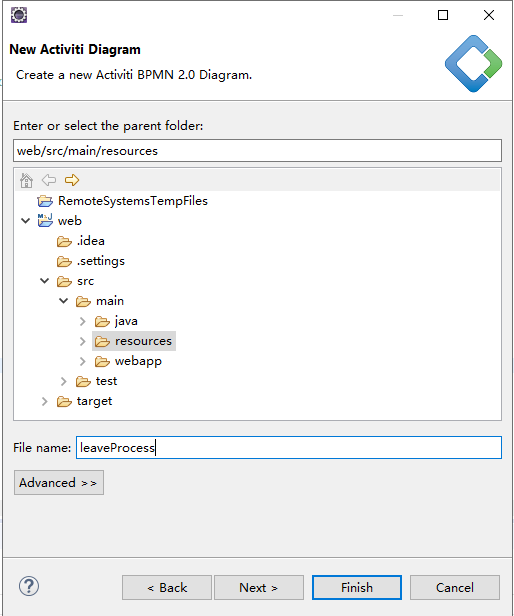


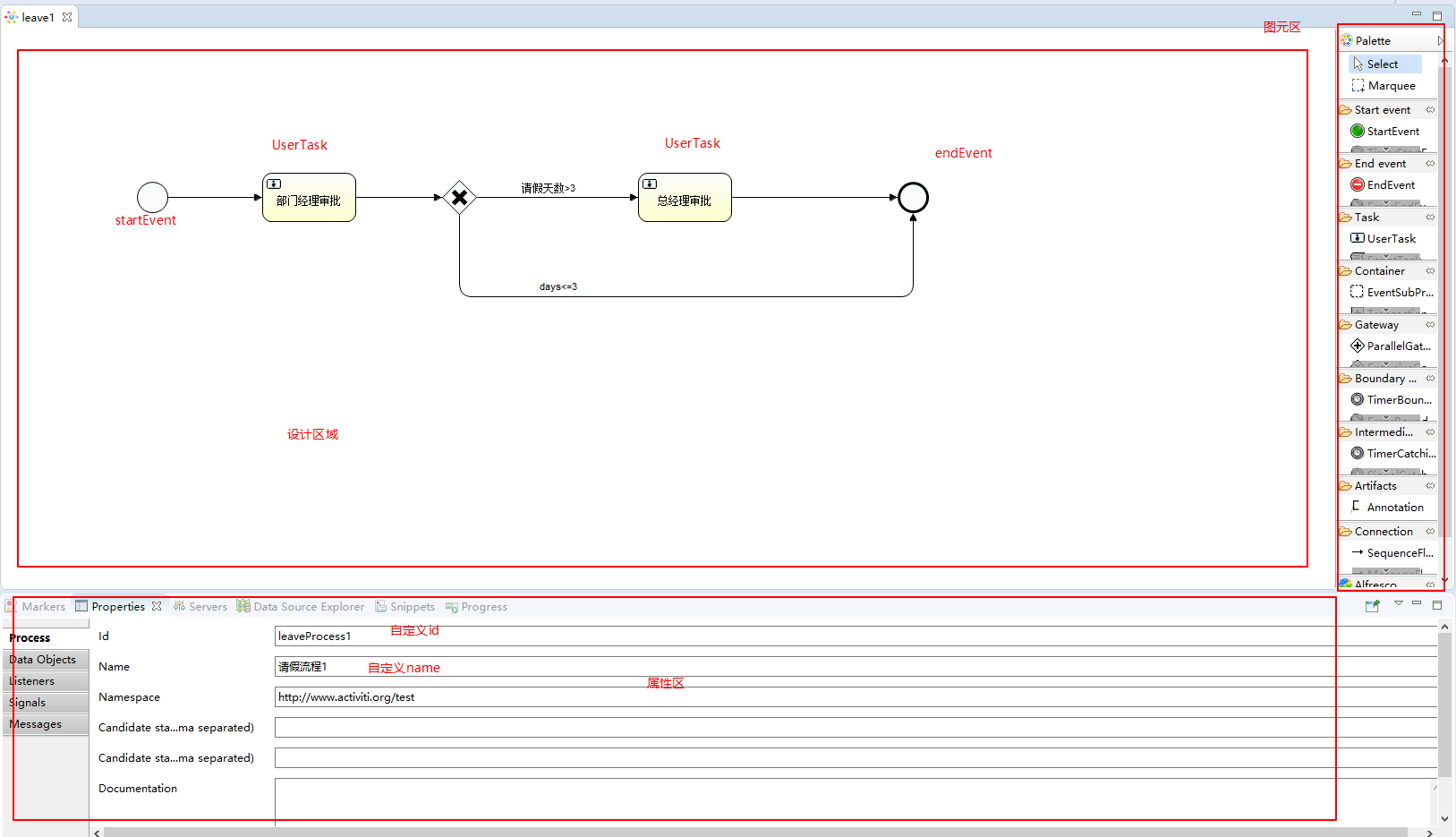
* 离线安装

在添加窗口点击Local按钮，选择本地磁盘下的插件文件夹。课件：15.activiti\designer插件。

插件安装成功后，在new创建下，选择如下：



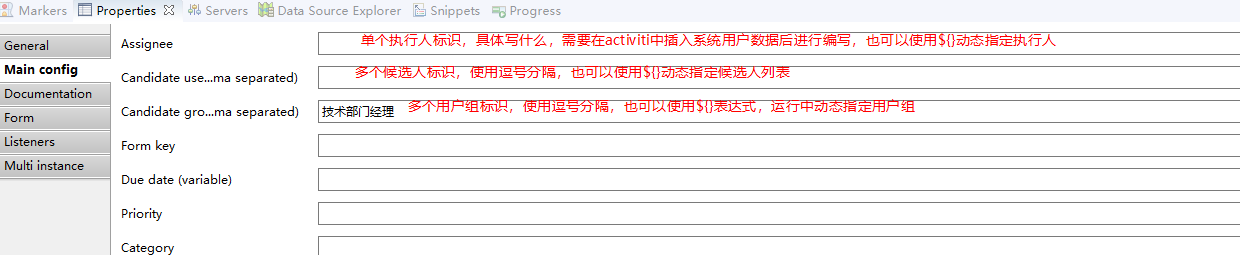




注意点1：id与name必须自定义为流程id与流程名，不能重复。

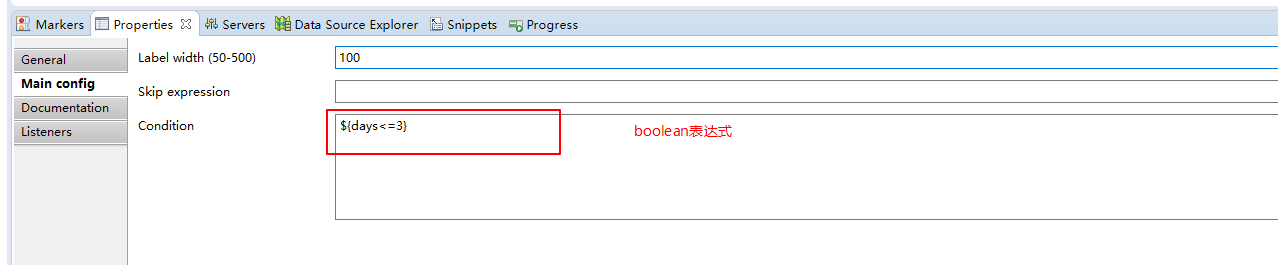
注意点2：一个流程中，有一个StartEvent，可以有多个endEvent；

注意点3：UsetTask中的Assignee,Candidate users,Candidate groups三者中选择一个指定任务参与人。

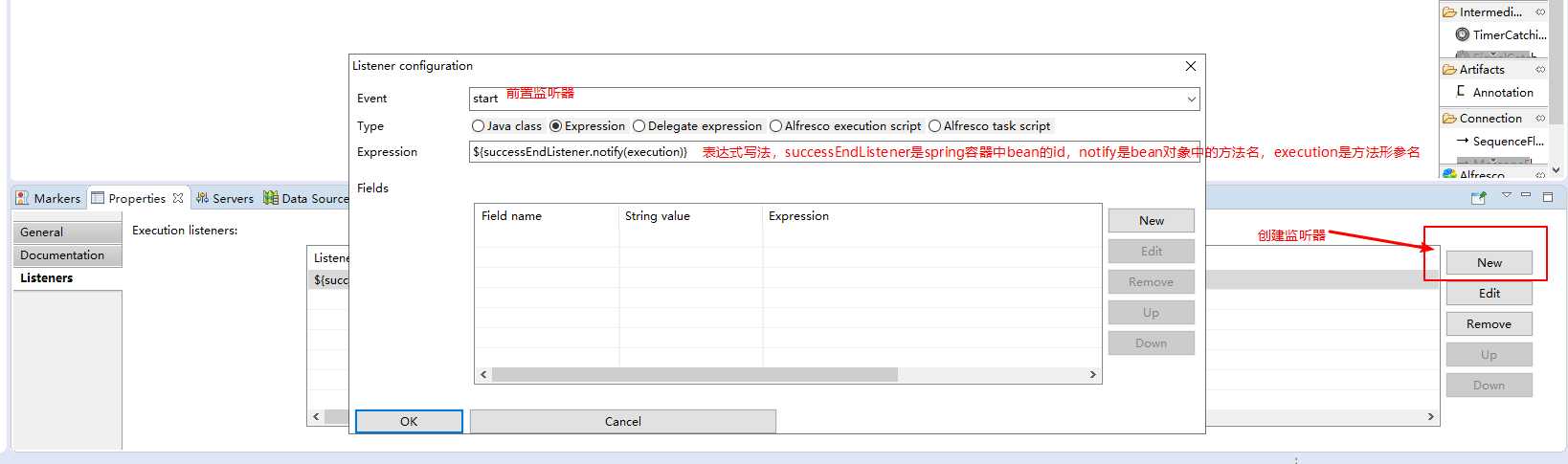


注意点4：当流程产生重要分支，需要加Gateway网关进行分支，多选一分支，使用ExclusiveGateway排它网关；并行分支，选用ParallelGateway并行网关。

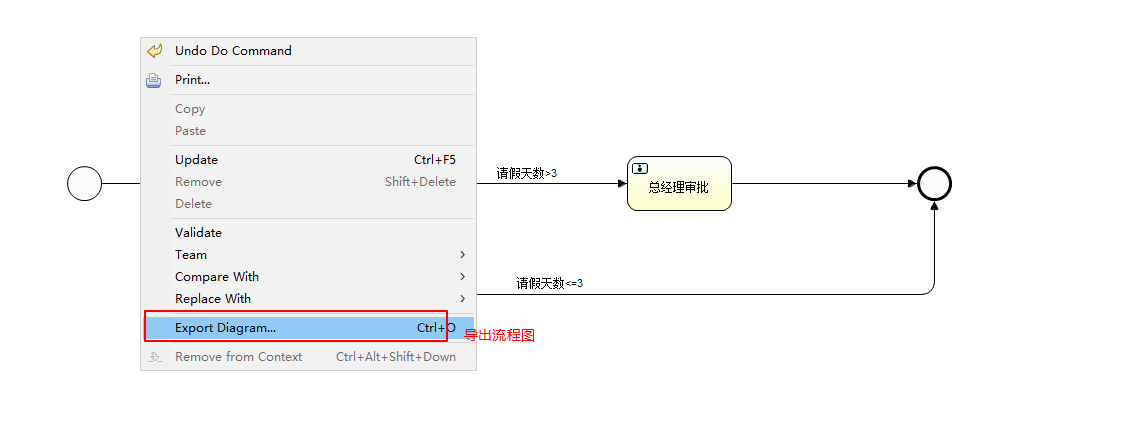
注意点5：分支后的connection连接线，需要指定分支条件。



注意点6：UserTask或endEvent，startEvent可以指定listener监听器，当执行到当前节点时，执行监听器代码，比如流程审批通过发邮件或短信通知申请人。以endEvent为例：



注意点7：流程设计完成后，导入图片。



### 代码运行环境

在ssm项目环境下安装activiti依赖包。

#### 添加maven依赖

|  |
| --- |
| <**dependency**>  <**groupId**>org.activiti</**groupId**>  <**artifactId**>activiti-engine</**artifactId**>  <**version**>5.22.0</**version**> </**dependency**> <**dependency**>  <**groupId**>org.activiti</**groupId**>  <**artifactId**>activiti-spring</**artifactId**>  <**version**>5.22.0</**version**> </**dependency**> |

#### 创建activiti的配置文件

在src/main/resources下创建activiti.xml，spring风格。

|  |
| --- |
| *<!--ProcessEngineConfiguration-->* <**bean id="processEngineConfiguration" class="org.activiti.spring.SpringProcessEngineConfiguration"**>  <**property name="dataSource" ref="dataSource"** />  <**property name="transactionManager" ref="transactionManager"** />  *<!--在创建ProcessEngine对象的时候是否创建更新数据库表-->* <**property name="databaseSchemaUpdate" value="false"** />  <**property name="jobExecutorActivate" value="false"** /> </**bean**>  *<!--ProcessEngine流程引擎核心对象-->* <**bean id="processEngine" class="org.activiti.spring.ProcessEngineFactoryBean"**>  <**property name="processEngineConfiguration" ref="processEngineConfiguration"** /> </**bean**> *<!--用来操作用户与角色信息-->* <**bean id="identityService" factory-bean="processEngine" factory-method="getIdentityService"**></**bean**> *<!--用来进行流程部署-->* <**bean id="repositoryService" factory-bean="processEngine" factory-method="getRepositoryService"** /> *<!--用来启动流程-->* <**bean id="runtimeService" factory-bean="processEngine" factory-method="getRuntimeService"** /> *<!--用来在流程启动后，执行任务签收，完成任务-->* <**bean id="taskService" factory-bean="processEngine" factory-method="getTaskService"** /> *<!--用来对于已结束的历史流程与历史任务查询-->* <**bean id="historyService" factory-bean="processEngine" factory-method="getHistoryService"** /> |

注意点：切记把activiti.xml的加载一定要配置。

## IdentityService服务

该服务用来管理引擎内依赖的用户与用户组数据。

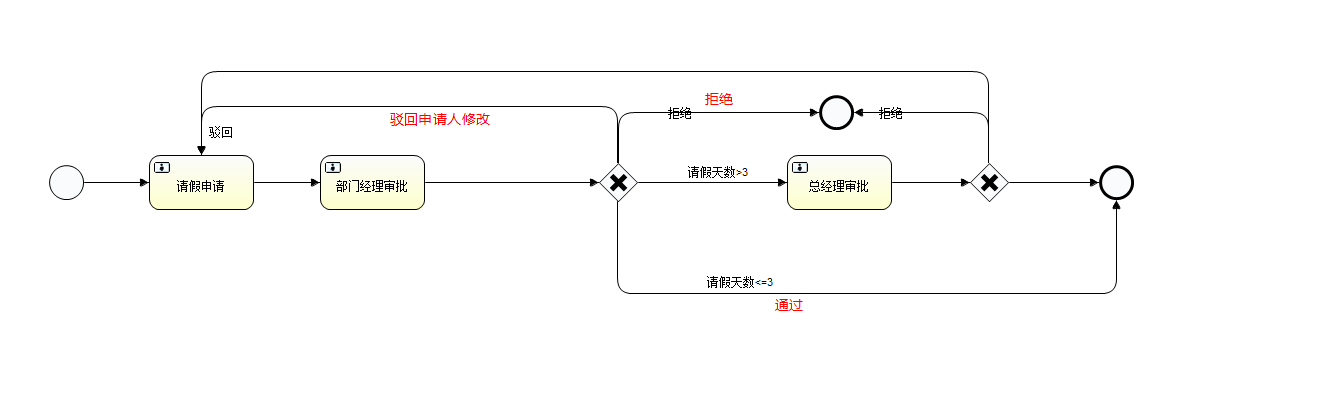
代码如下：

|  |
| --- |
| */\*\*  \* 用户与用户组数据不能删除，不能修改id  \*/* @RunWith(SpringJUnit4ClassRunner.**class**) @ContextConfiguration(**"classpath:spring.xml"**) **public class** TestIdentityService {  @Resource  **private** IdentityService **ds**;   *//添加引擎依赖用户，在系统管理下用户管理功能处加入如下代码，维护两套用户数据。* @Test  **public void** test1\_addUser(){  *//用户唯一标识（登录账号）* User user1 = **ds**.newUser(**"jpc"**);  User user2 = **ds**.newUser(**"zbw"**);  User user3 = **ds**.newUser(**"lhd"**);  User user4 = **ds**.newUser(**"wts"**);  **ds**.saveUser(user1);  **ds**.saveUser(user2);  **ds**.saveUser(user3);  **ds**.saveUser(user4);  }   *//添加引擎依赖用户组，在角色管理模块处加入如下代码，维护两套用户组数据* @Test  **public void** test1\_addGroup(){  Group g1 = **ds**.newGroup(**"普通员工"**);  Group g2 = **ds**.newGroup(**"技术部门经理"**);  Group g3 = **ds**.newGroup(**"总经理"**);  **ds**.saveGroup(g1);  **ds**.saveGroup(g2);  **ds**.saveGroup(g3);  }   *//为用户分配用户组* @Test  **public void** test1\_memberShip(){  **ds**.createMembership(**"jpc"**,**"普通员工"**);  **ds**.createMembership(**"zbw"**,**"技术部门经理"**);  **ds**.createMembership(**"lhd"**,**"技术部门经理"**);  **ds**.createMembership(**"wts"**,**"总经理"**);  } } |

注意点：在流程设计时，为UserTask指定的执行人候选人，用户组数据都是这里维护的数据标识。

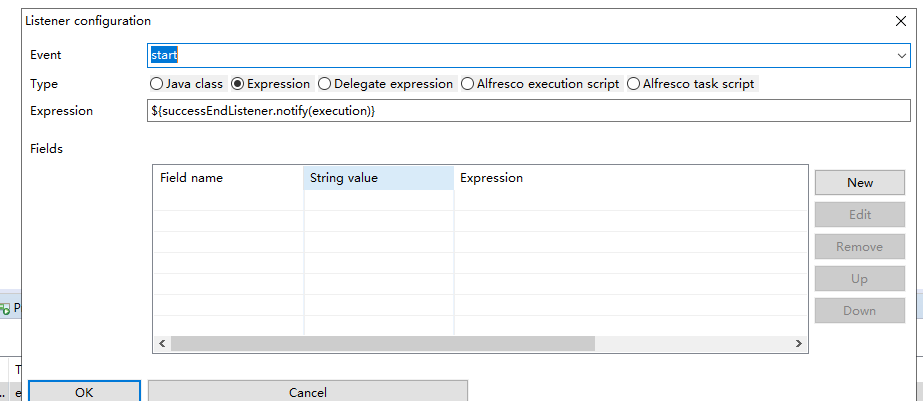
## 流程设计

假设审批人有三种审批结果：通过，驳回，拒绝。

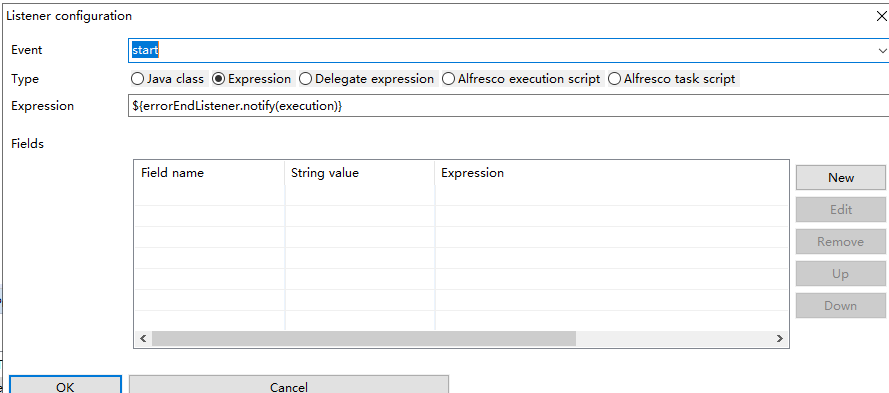


注意点1：UserTask一定要指定执行人或候选人或候选用户组。

注意点2：这里为两个endEvent分别指定了Listener，成功结束的如下：



拒绝的如下：



监听器代码如下：

|  |
| --- |
| @Service(**"successEndListener"**) **public class** SuccessEndListener {  @Resource  **private** ILeaveService **ls**;   **public void** notify(DelegateExecution execution){  String processInstanceId = execution.getProcessInstanceId();  System.***out***.println(**"审核通过结束："**+processInstanceId);  *//把请假申请单的状态修改为审核通过* **ls**.updateStatusByProcessId(processInstanceId,**"审批通过"**);  } } |

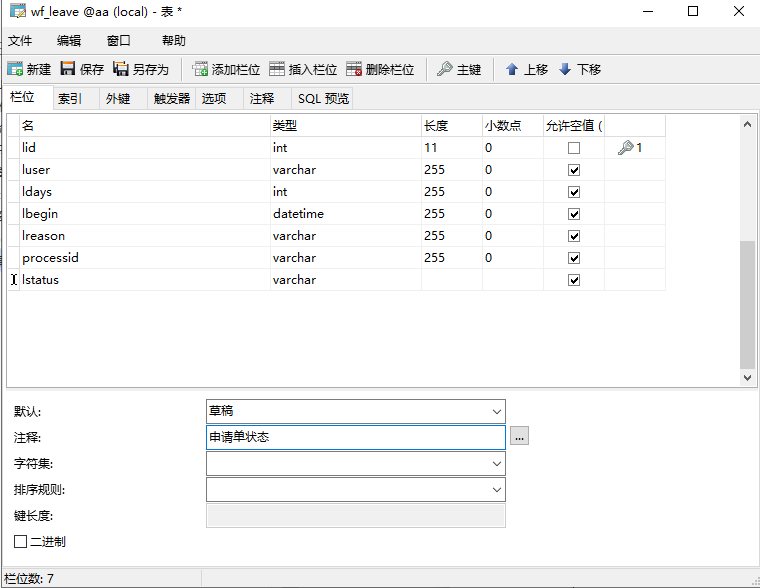
|  |
| --- |
| @Service(**"errorEndListener"**) **public class** ErrorEndListener {  @Resource  **private** ILeaveService **ls**;   **public void** notify(DelegateExecution execution){  String processInstanceId = execution.getProcessInstanceId();  System.***out***.println(**"审核不通过结束："**+processInstanceId);  *//把请假申请单的状态修改为审核未通过* **ls**.updateStatusByProcessId(processInstanceId,**"审批未通过"**);  } } |

## RepositoryService服务

该服务用来管理流程部署，把设计的流程定义文件部署到数据库。

|  |
| --- |
| @RunWith(SpringJUnit4ClassRunner.**class**) @ContextConfiguration(**"classpath:spring.xml"**) **public class** TestRepositoryService {   @Resource  **private** RepositoryService **rs**;   *//部署流程* @Test  **public void** deploy(){  DeploymentBuilder deployment = **rs**.createDeployment();  deployment.addClasspathResource(**"workflow/leave2.bpmn"**);  deployment.addClasspathResource(**"workflow/leave2.png"**);  deployment.name(**"请假流程2部署"**);  Deployment deploy = deployment.deploy();*//向数据库中添加流程定义文件* System.***out***.println(deploy.getId());*//2501* }    *//查询流程部署记录* @Test  **public void** queryDeploy(){  DeploymentQuery query = **rs**.createDeploymentQuery();  List<Deployment> list = query.list();  }   *//删除流程部署* @Test  **public void** del(){  **rs**.deleteDeployment(**"30001"**);  } } |

## 创建请假申请单业务表与实体类



对应实体类

|  |
| --- |
| **public class** WfLeave {  **private** Integer **lid**;  **private** String **luser**;  **private** Integer **ldays**;  **private** String **lreason**;  **private** String **lbegin**;  **private** String **processid**; *//流程实例id* **private** String **lstatus;**  *//审批状态,默认草稿状态*  *}* |

对应的service层与dao层基本curd代码，省略。

## RuntimeService服务

启动流程

|  |
| --- |
| */\*\*  \* 提交流程审批  \** ***@param leave,请假申请单对象****，该对象数据要完整，有id，days，luser信息  \*/* @Override **public void** startProcess(WfLeave leave) {  *//1.把请假申请单保存到请假申请表，默认草稿状态* **this**.addLeaveApply(leave);  *//2. 启动流程，"leaveProcess2"是流程设计时，指定的id,days与loginUser是流程设计时需要的流程变量* Map<String,Object> vali = **new** HashMap<>();  vali.put(**"days"**,leave.getLdays());  vali.put(**"loginUser"**,leave.getLuser());  ProcessInstance procInst = **rs**.startProcessInstanceByKey(**"leaveProcess2"**, vali);*//流程实例对象  //新启动的流程实例id* String procId = procInst.getId();  *//把ProcessId更新到业务表中的processid外键* **this**.updateProcessId2Leave(procId,leave.getLid()+**""**);  *//把请假申请节点走完* List<Task> todoTask = **this**.getTodoTask(leave.getLuser());  String taskid = todoTask.get(0).getId();  **this**.completeTask(taskid,**null**); }  @Override **public void** updateProcessId2Leave(String processId, String leaveId) {  WfLeave l = **new** WfLeave();  l.setLid(Integer.*parseInt*(leaveId));  l.setProcessid(processId);  l.setLstatus(**"审批中"**);  **lm**.updateByPrimaryKeySelective(l); }  @Override **public void** updateStatusByProcessId(String processInstanceId, String result) {  **lm**.updateStatusByProcessId(processInstanceId,result); } |

## TaskService服务

### 查询待办任务

|  |
| --- |
| @Override  **public** List<Task> getTodoTask(String loginUserId) {  TaskQuery taskQuery = **ts**.createTaskQuery();  taskQuery.taskCandidateOrAssigned(loginUserId);  *//如果不指定查询流程定义的key。则查询所有业务的待办任务,适合于用在工作台欢迎页面* taskQuery.processDefinitionKey(**"leaveProcess2"**);*//这里表示查询请假审批的待办任务 // long count = taskQuery.count();//查询总记录数 // taskQuery.listPage(1,3);//分页查询* List<Task> list = taskQuery.list();  **return** list;  } |

### 多人任务签收

|  |
| --- |
| @Override **public void** doClaim(String loginUser, String taskid) {  **ts**.claim(taskid,loginUser); } |

### 签收后审批

|  |
| --- |
| */\*\*  \*   \** ***@param taskid,审批任务id*** *\** ***@param result****：审批结果(通过，拒绝，驳回)  \** ***@param comment****:审批意见  \*/* @Transactional @Override **public void** doApprove(String taskid, String result, String comment) {  TaskQuery taskQuery = **ts**.createTaskQuery();  taskQuery.taskId(taskid);  Task task = taskQuery.singleResult();  String processInstanceId = task.getProcessInstanceId();  **ts**.addComment(taskid,processInstanceId,result,comment);  **this**.completeTask(taskid,result); }  @Override **public void** completeTask(String taskid,String result) {  **if**(result==**null**){  **ts**.complete(taskid);  **return**;  }   Map<String,Object> vali = **new** HashMap<>();  *//1：通过，2：不通过，3：驳回* vali.put(**"approveResult"**,result);  **ts**.complete(taskid,vali); } |

## HistoryService流程跟踪

创建实体

|  |
| --- |
| **public class** HistoryTaskAndComment {  **private** String **taskid**;*//任务id* **private** String **processId**;*//流程实例id* **private** String **taskName**;*//任务名* **private** String **creatTime**;*//任务开始时间* **private** String **endTime**;*//结束时间* **private** String **result**;*//审批结果* **private** String **comment**;*//审批意见* **private** String **assignee**;*//任务执行人*  *}* |

### 申请人查询历史任务

|  |
| --- |
| */\*\*  \* 作为申请人，查看申请单下的已执行节点，以及任务审批意见  \** ***@param processId*** *\** ***@return*** *\*/* @Override **public** List<HistoryTaskAndComment> getHistoryTaskByProcessId(String processId) {  List<HistoryTaskAndComment> result = **new** ArrayList<>();  *//得到执行流程实例下的所有审批意见* List<Comment> comments = **ts**.getProcessInstanceComments(processId);  *//查询历史任务* HistoricTaskInstanceQuery htq = **hs**.createHistoricTaskInstanceQuery();*//历史任务查询* htq.processInstanceId(processId);  htq.orderByTaskCreateTime();  htq.asc();    *//把历史任务和审批意见组合* List<HistoricTaskInstance> list = htq.list();  **for** (HistoricTaskInstance t : list) {  HistoryTaskAndComment tc = **new** HistoryTaskAndComment(t.getId(),t.getProcessInstanceId(),t.getName(),DateUtils.*formatYMDHmS*(t.getStartTime()),DateUtils.*formatYMDHmS*(t.getEndTime()),t.getAssignee());  Comment c = **this**.getCommentsByTaskId(comments,t.getId());  **if**(c!=**null**)  {  tc.setResult(c.getType());  tc.setComment(c.getFullMessage());  }  result.add(tc);  }  **return** result; }  **private** Comment getCommentsByTaskId(List<Comment> comments,String taskId){  **for** (Comment comment : comments) {  **if** (taskId.equals(comment.getTaskId())){  **return** comment;  }  }  **return null**; } |

### 审批人查询已办任务

|  |
| --- |
| */\*\*  \* 某节点审批人登录系统，查询自身已办的任务  \** ***@param loginuserId*** *\*/* @Override **public void** getHistoryTaskByAssignee(String loginuserId) {  HistoricTaskInstanceQuery htq = **hs**.createHistoricTaskInstanceQuery();  htq.taskAssignee(loginuserId);  htq.processDefinitionKey(***key2***);  List<HistoricTaskInstance> list = htq.list();  **for** (HistoricTaskInstance t : list) {  System.***out***.println(t.getId()+**"-"**+t.getProcessInstanceId()+**"-"**+t.getAssignee()+**"-"**+t.getEndTime()+**"-"**+t.getName());  } } |