Activity学习以及使用

## 1.Activiti介绍

### 1.1.Activiti相关网站

Activiti官网 : <https://www.activiti.org/>

Activiti的一个中文手册: <http://www.mossle.com/docs/activiti/index.html#N133A6>

## 2.安装Activiti相关插件

### 2.1.Eclipse安装Activiti插件

 解压这个文件,把features和plugins两个文件夹中的文件复制到eclipse安装目录的对应的features和plugins文件夹中,重启即可.

### 2.2.IDEA安装Activiti插件

<https://blog.csdn.net/qq_41728540/article/details/79506463>

## 3.如何使用(java Maven项目测试)

### 3.1.Activiti的自动建表

//创建流程引擎 并且没有表的话创建流程表

ProcessEngine processEngine = ProcessEngineConfiguration.createProcessEngineConfigurationFromResource("activiti.cfg.xml").buildProcessEngine();

#### 3.1.1.Activiti的表的结构



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。

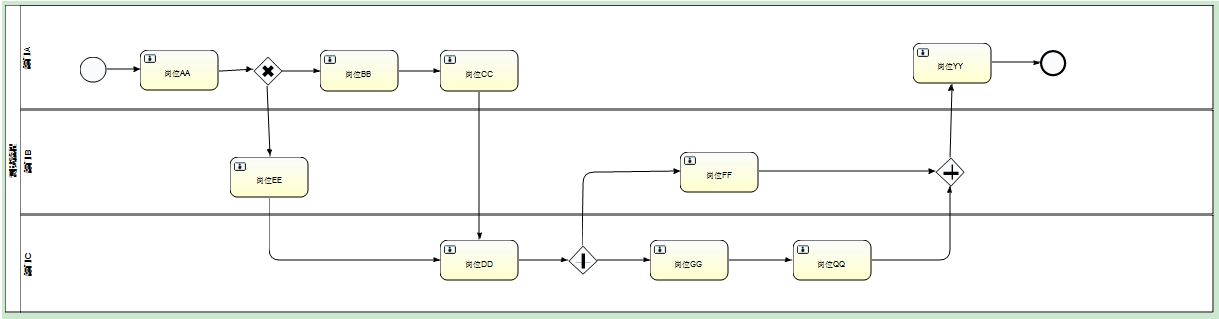
ACT\_ID\_\*: 'ID'表示identity。 这些表包含身份信息，比如用户，组等等。

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。

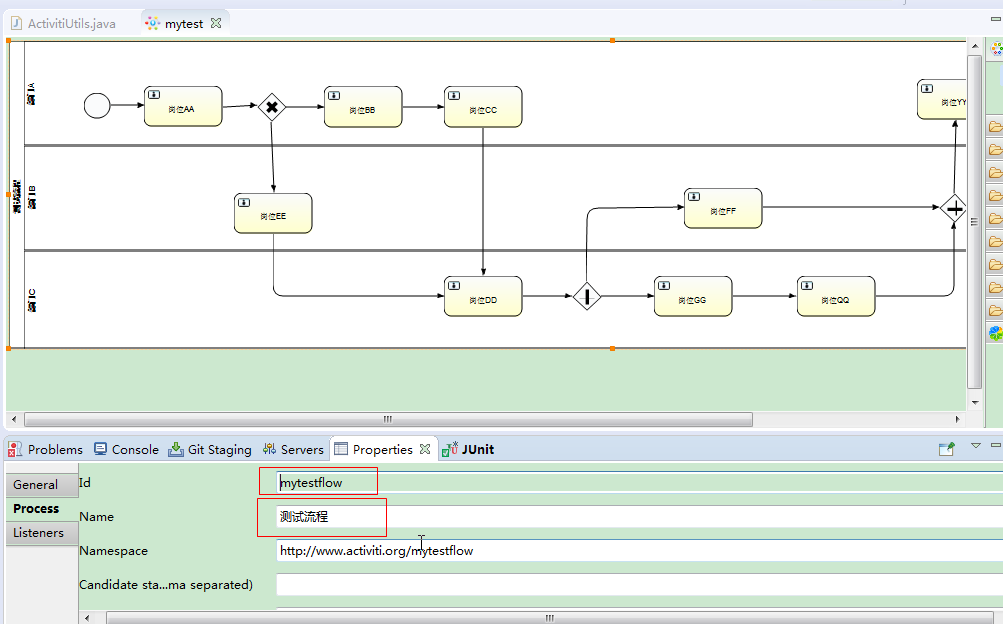
ACT\_GE\_\*: 通用数据， 用于不同场景下。

### 3.2.画流程图

Idea和eclipse都有插件可以画,但是eclipse的支持性好些.idea的插件对话流程图的池子和泳道不太好. 池子和泳道不会影响逻辑,但是看起来更清晰.池子一般代表一个流程,泳道一般代表一个组织或者部门,相应的岗位在这个组织或者部门中,方便理解.

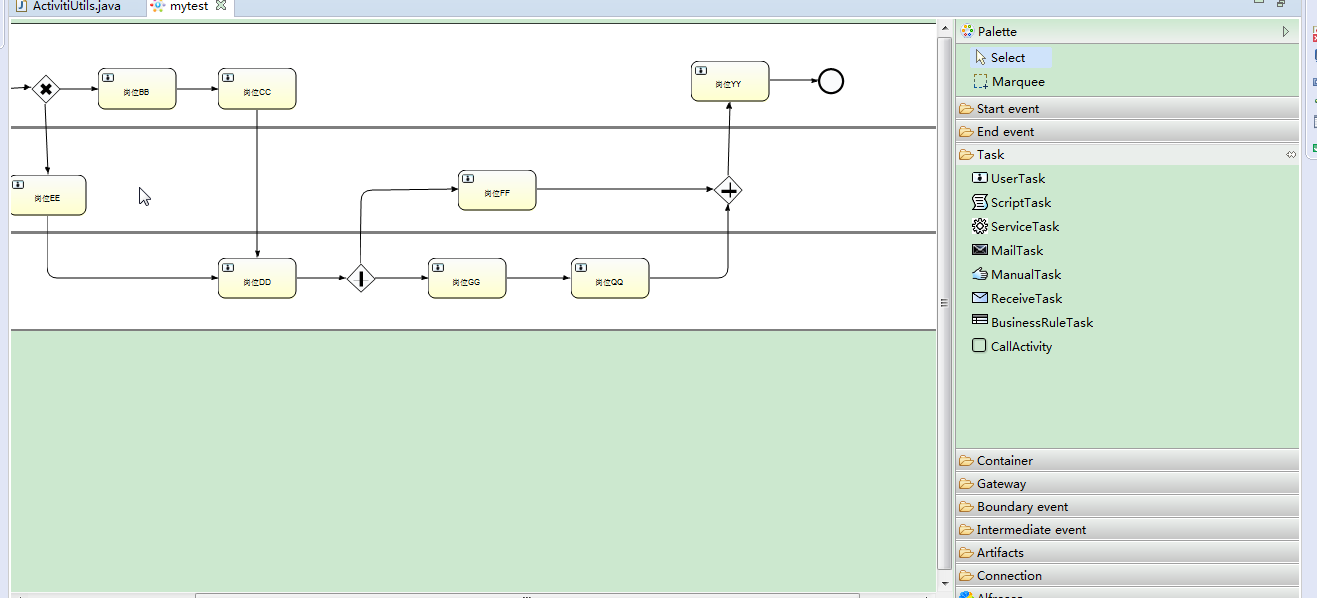


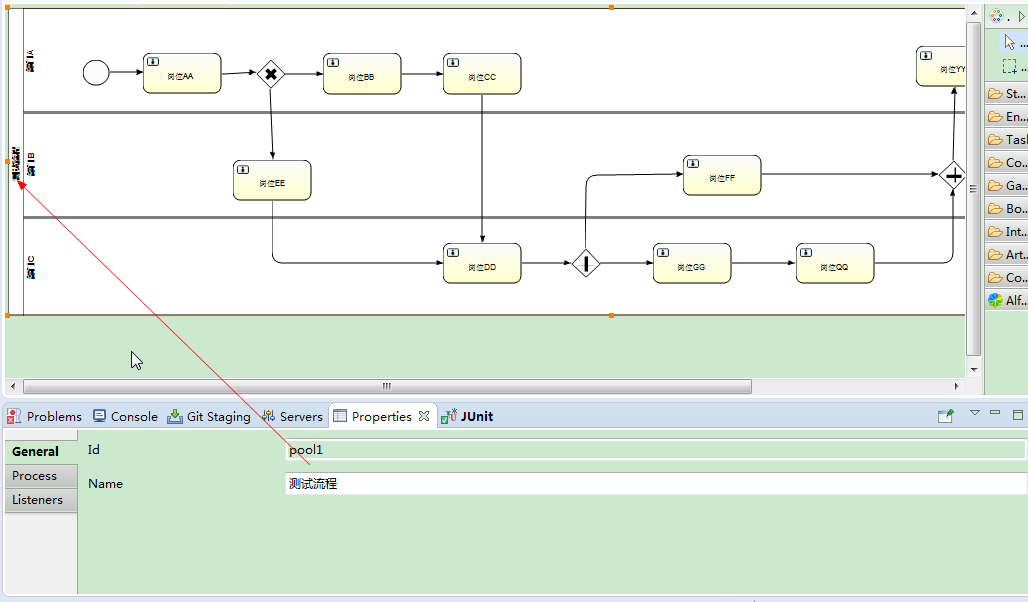
定义流程id和名字



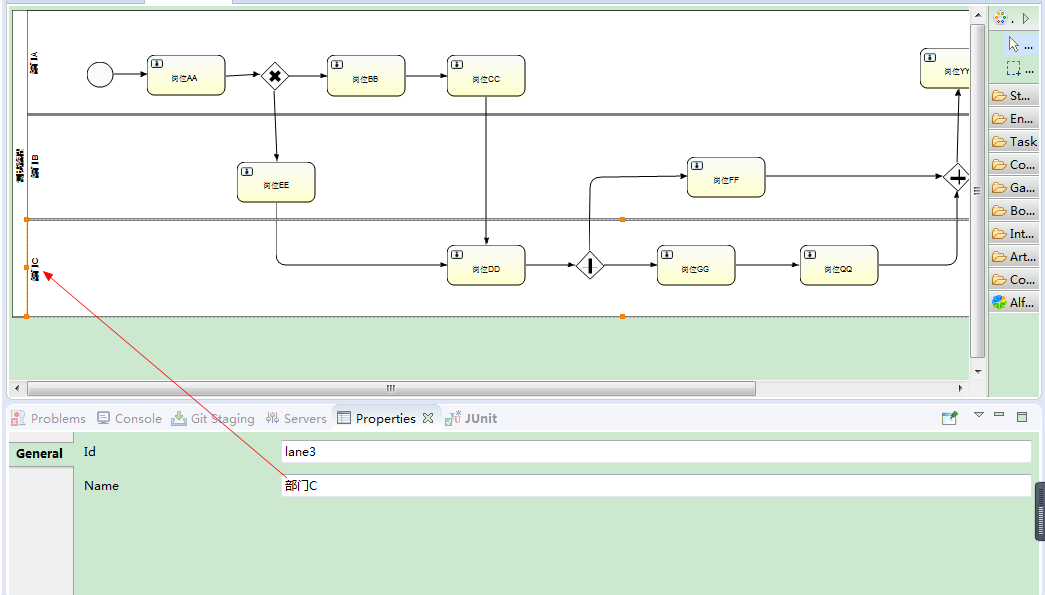
定义池子的id和name

拖拽实现创建

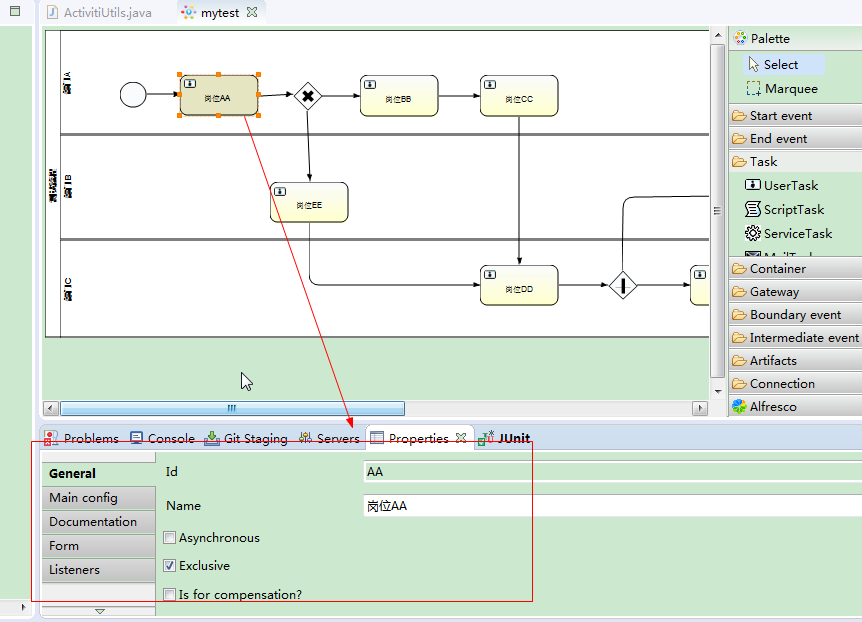




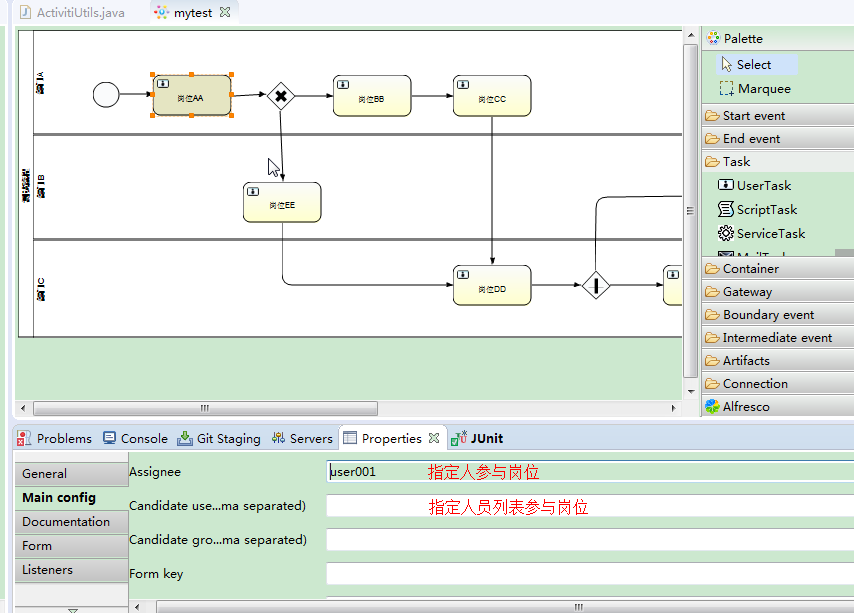
定义泳道的id和name

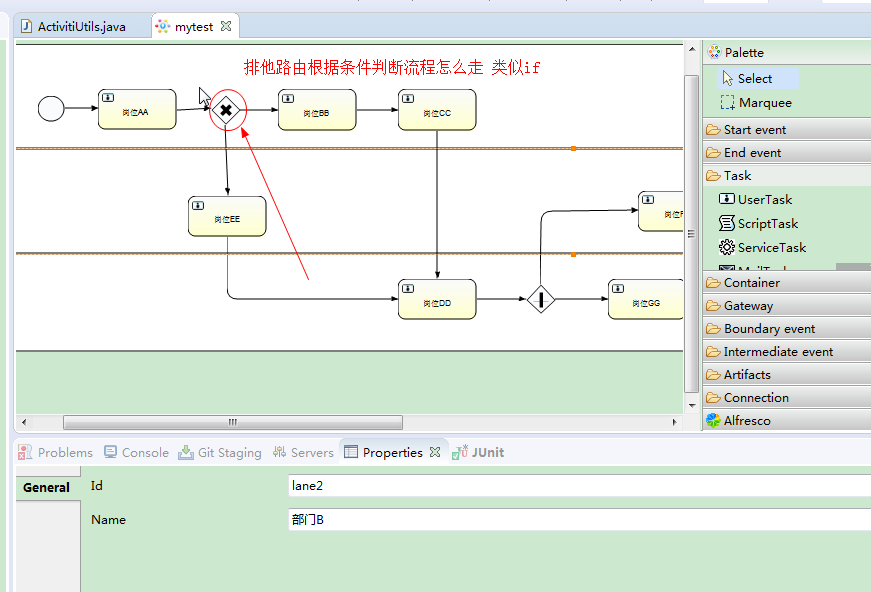


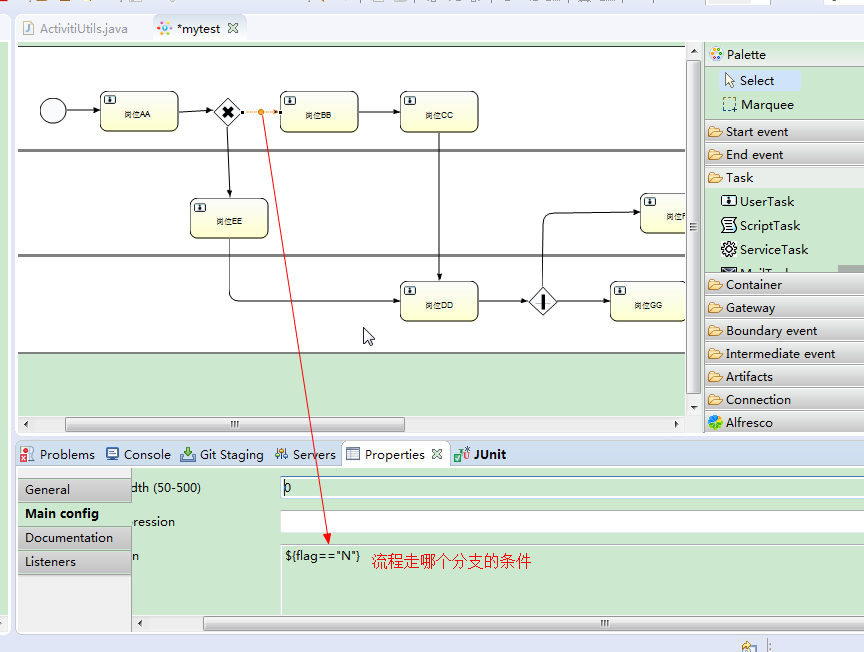
岗位指定name和id还有人员列表

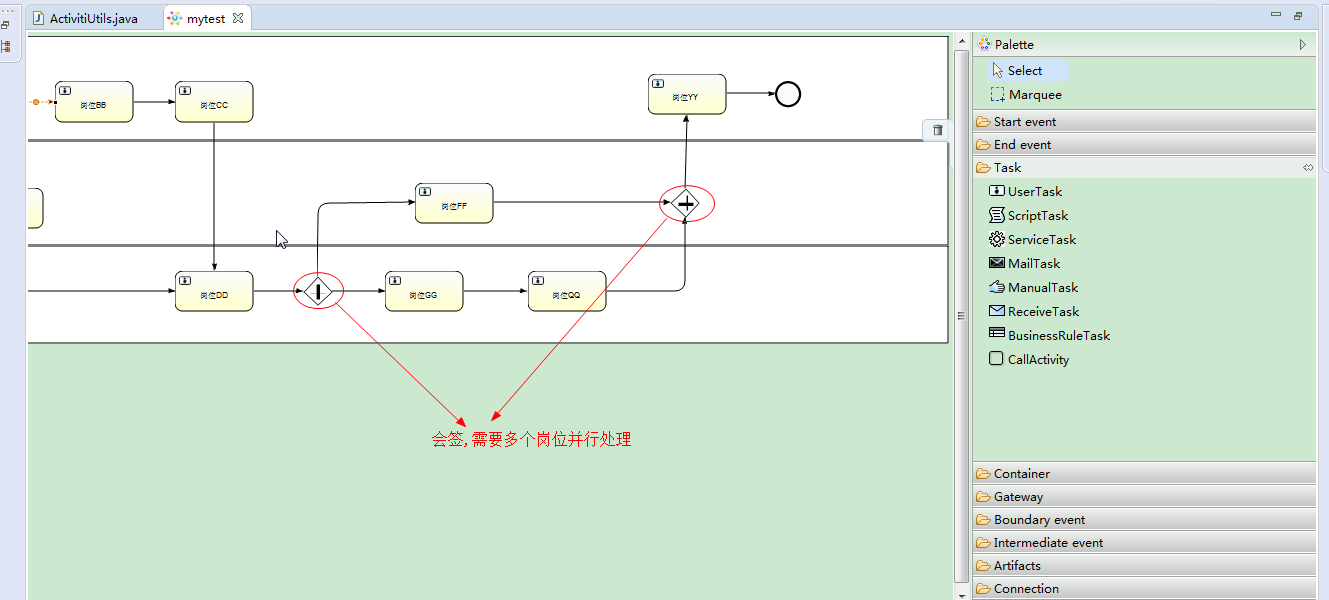


Assignee不用签收任务直接就到的对应的人的名下









### 3.3.写代码走流程

#### 3.3.1.部署流程

//部署指定的流程

RepositoryService repositoryService = processEngine.getRepositoryService();

DeploymentBuilder builder = repositoryService.createDeployment();

builder.addClasspathResource("mytest.bpmn");//bpmn文件的名称

Deployment deploy = builder.deploy();

#### 3.3.1.查询部署的流程

ProcessDefinitionQuery processDefinitionQuery = processEngine.getRepositoryService().createProcessDefinitionQuery();

processDefinitionQuery.processDefinitionKey("mytestflow").latestVersion();

List<ProcessDefinition> list = processDefinitionQuery.list();

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

System.***out***.println(processDefinition.getId()+"-"+processDefinition.getName()+"-"+processDefinition.getVersion());

}

#### 3.3.2.启动部署的流程

RuntimeService runtimeService = processEngine.getRuntimeService();

ProcessInstance processInstance = runtimeService.startProcessInstanceById("mytestflow:1:4");

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

//挂起流程实例

//processEngine.getRuntimeService().suspendProcessInstanceById("17501");

//激活流程实例

// processEngine.getRuntimeService().activateProcessInstanceById("17501");

//删除流程实例

//processEngine.getRuntimeService().deleteProcessInstance("17501","删除原因");

#### 3.3.2.查询当前用户未签收

//TaskQuery toClaimQuery = processEngine.getTaskService().createTaskQuery().taskCandidateUser("M001").active();

//List<Task> list = toClaimQuery.list();

//for (Task task : list) {

// System.out.println(task.getId());

//}

#### 3.3.3.签收任务

//签收任务

// processEngine.getTaskService().claim("5004","M003");

//查询已签收的任务

// TaskQuery todoTaskQuery = processEngine.getTaskService().createTaskQuery().taskAssignee("user001").active();

#### 3.3.4.办理任务

// HashMap<String, Object> hashMap = new HashMap<>();

// hashMap.put("flag","Y");

// //办理任务

// processEngine.getTaskService().complete("10002",hashmap);

#### 3.3.5.流程变量

//获取流程变量 常用的对象都是可以去设置获取流程变量的依赖于流程 类似 treadlocal

ProcessInstance processInstance = runtimeService

.startProcessInstanceByKey(processDefinitionKey, variables);

taskService.complete(taskId, variables);

runtimeService.setVariable("503", "price", 10000);

runtimeService.setVariable("503", "price", 10000);

/\*Object value1 = processEngine.getTaskService().getVariable("27502", "key1");

Map<String, Object> variables = processEngine.getTaskService().getVariables("27502");

Map<String, Object> variables1 = processEngine.getRuntimeService().getVariables("27501");\*/

#### 3.3.6.流程回退(中国特色)

**public** **static** **void** TaskRollBack(String taskId){

**try** {

Map<String, Object> variables;

// 取得当前任务

HistoricTaskInstance currTask = *historyService*

.createHistoricTaskInstanceQuery().taskId(taskId)

.singleResult();

// 取得流程实例

ProcessInstance instance = *runtimeService*

.createProcessInstanceQuery()

.processInstanceId(currTask.getProcessInstanceId())

.singleResult();

**if** (instance == **null**) {

//流程结束

}

variables = instance.getProcessVariables();

// 取得流程定义

ProcessDefinitionEntity definition = (ProcessDefinitionEntity) (*processEngine*.getRepositoryService().getProcessDefinition(currTask

.getProcessDefinitionId()));

**if** (definition == **null**) {

//log.error("流程定义未找到");

**return** ;

}

// 取得上一步活动

ActivityImpl currActivity = ((ProcessDefinitionImpl) definition)

.findActivity(currTask.getTaskDefinitionKey());

//可以修改currActivity的值为指定的一个岗位就可以跳转到指定的流程岗位

//进入当前节点的进入的线

List<PvmTransition> nextTransitionList = currActivity

.getIncomingTransitions();

// 清除当前活动的出口

List<PvmTransition> oriPvmTransitionList = **new** ArrayList<PvmTransition>();

List<PvmTransition> pvmTransitionList = currActivity.getOutgoingTransitions();

**for** (PvmTransition pvmTransition : pvmTransitionList) {

oriPvmTransitionList.add(pvmTransition);

}

pvmTransitionList.clear();

// 建立新出口

List<TransitionImpl> newTransitions = **new** ArrayList<TransitionImpl>();

**for** (PvmTransition nextTransition : nextTransitionList) {

PvmActivity nextActivity = nextTransition.getSource();

ActivityImpl nextActivityImpl = ((ProcessDefinitionImpl) definition)

.findActivity(nextActivity.getId());

TransitionImpl newTransition = currActivity

.createOutgoingTransition();

newTransition.setDestination(nextActivityImpl);

newTransitions.add(newTransition);

}

// 完成任务

List<Task> tasks = *taskService*.createTaskQuery()

.processInstanceId(instance.getId())

.taskDefinitionKey(currTask.getTaskDefinitionKey()).list();

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

*taskService*.complete(task.getId(), variables);

//删除当前节点(还是留痕好一些)

// historyService.deleteHistoricTaskInstance(task.getId());

}

// 恢复方向

**for** (TransitionImpl transitionImpl : newTransitions) {

currActivity.getOutgoingTransitions().remove(transitionImpl);

}

**for** (PvmTransition pvmTransition : oriPvmTransitionList) {

pvmTransitionList.add(pvmTransition);

}

**return** ;

} **catch** (Exception e) {

**return** ;

}

}

#### 3.3.7.流程图展示

@RequestMapping("showFlowImg")

**public** **void** showFlowImg(HttpServletResponse response, String processDefinitionId) **throws** IOException {

BpmnModel bpmnModel = repositoryService.getBpmnModel(processDefinitionId);

List<String> activeActivityIdList = **new** ArrayList<>();

ArrayList<String> highLightedFlowList = **new** ArrayList<>();

ProcessDiagramGenerator processDiagramGenerator = processEngineConfiguration.getProcessDiagramGenerator();

InputStream imageStream = processDiagramGenerator.generateDiagram(bpmnModel, "png", activeActivityIdList,

highLightedFlowList, "宋体", "宋体", **null**, 1.0);

BufferedInputStream bufferedInputStream = **new** BufferedInputStream(imageStream);

BufferedOutputStream bufferedOutputStream = **new** BufferedOutputStream(response.getOutputStream());

**int** a;

**while** ((a = bufferedInputStream.read()) != -1) {

bufferedOutputStream.write(a);

}

bufferedInputStream.close();

bufferedOutputStream.close();

}

#### 3.3.8.流程图展示(当前任务高亮)

@RequestMapping("showActiveTaskInFlowImg")

**public** **void** showActiveTaskInFlowImg(HttpServletResponse response,String processInstanceId) **throws** IOException {

ProcessInstance processInstance = runtimeService.createProcessInstanceQuery().processInstanceId(processInstanceId).singleResult();

BpmnModel bpmnModel = repositoryService.getBpmnModel(processInstance.getProcessDefinitionId());

List<String> activeActivityIdList = runtimeService.getActiveActivityIds(processInstanceId);

ProcessDiagramGenerator processDiagramGenerator = processEngineConfiguration.getProcessDiagramGenerator();

ArrayList<String> highLightedFlowList = **new** ArrayList<>();

InputStream imageStream = processDiagramGenerator.generateDiagram(bpmnModel, "png", activeActivityIdList,

highLightedFlowList, "宋体", "宋体", **null**, 1.0);

BufferedInputStream bufferedInputStream = **new** BufferedInputStream(imageStream);

BufferedOutputStream bufferedOutputStream = **new** BufferedOutputStream(response.getOutputStream());

**int** a;

**while** ((a = bufferedInputStream.read()) != -1) {

bufferedOutputStream.write(a);

}

bufferedInputStream.close();

bufferedOutputStream.close();

}

## 4.Activiti的常用的对象

这个框架提供了非常丰富的api 基本操作都是ok的

基本都是链式编程,还有就是有什么问题在网上搜索不叫方便.

* 几个和流程相关的对象

Deployment:部署对象，和部署表对应act\_re\_deployment

ProcessDefinition:流程定义对象,和流程定义表对应act\_re\_procdef

ProcessInstance:流程实例对象,和流程实例表对应act\_ru\_execution

Task:任务对象,和任务表对应act\_ru\_task

* 几个Service对象

RepositoryService:操作部署、流程定义等静态资源信息

RuntimeService:操作流程实例，启动流程实例、查询流程实例、删除流程实例等动态信息

TaskService:操作任务，查询任务、办理任务等和任务相关的信息

HistoryService:操作历史信息的，查询历史信息

IdentityService:操作用户和组

* 几个Query对象

DeploymentQuery:对应查询部署表act\_re\_deployment

ProcessDefinitionQuery:对应查询流程定义表act\_re\_procdef

ProcessInstanceQuery:对应查询流程实例表act\_ru\_execution

TaskQuery:对应查询任务表act\_ru\_task

代码



常用的几个表

select \* from act\_re\_deployment; #流程部署

select \* from act\_re\_procdef; #流程定义

select \* from act\_ru\_execution; #流程实例

select \* from act\_ru\_task; #流程任务

select \* from act\_ru\_variable; #流程变量

select \* from act\_hi\_procinst; #历史流程实例

select \* from act\_hi\_varinst; #历史流程变量

select \* from act\_hi\_taskinst ORDER BY START\_TIME\_ desc;#历史流程任务