# Activiti工作流开发笔记

**待研究：**

**IdentityService**

**基本步骤：**

## 1.业务分析，设计流程图

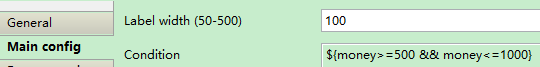
流程图任务节点id 使用\*\_task后缀作为名称，方便程序后续的绑定：



任务节点处理人：



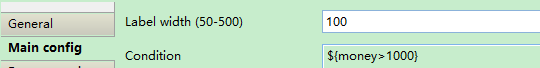
分支条件设置示例  
  1.500≤费用金额≤1000



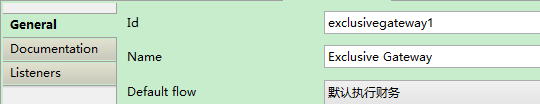
     2.费用金额＜500

http://img.blog.csdn.net/20151220154502048

     3.费用金额＞1000



     4.设置排他网关的默认执行任务



     5.设置【财务】审批为默认

http://img.blog.csdn.net/20151220154935907

## 2.部署流程定义

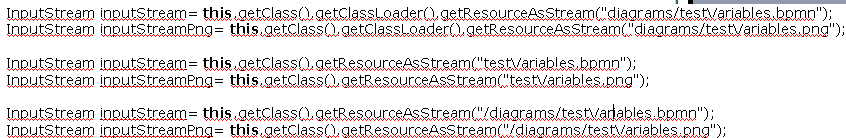
#### 概要：

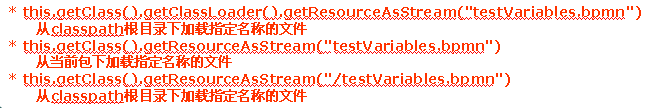
初步设想实现一个部署完工作流之后，为UserTask节点动态分配任务执行者，或者在分支节点上添加进入条件的功能。为了实现这个功能需要解析流程定义文件取出文件中定义的所有节点。

#### 部署流程定义



输入流加载资源文件的3种方式





#### 设计用户节点审核表

动态加载节点，为节点绑定审核人

#### 删除流程定义

* + - 1. /\*
      2. \* 删除流程定义
      3. \*/
      4. @Test
      5. public void deleteProcessDefinition(){
      6. //使用部署ID，完成删除
      7. String deploymentId = "1";
      8. /\*
      9. \* 不带级联的删除
      10. \* 只能删除没有启动的流程，如果流程启动，就会抛出异常
      11. \*/
      12. // processEngine.getRepositoryService()//与流程定义和部署对象相关的Service
      13. // .deleteDeployment(deploymentId);
      14. /\*
      15. \* 能级联的删除
      16. \* 能删除启动的流程，会删除和当前规则相关的所有信息，正在执行的信息，也包括历史信息
      17. \*/
      18. processEngine.getRepositoryService()//与流程定义和部署对象相关的Service
      19. .deleteDeployment(deploymentId, true);
      21. System.out.println("删除成功");
      22. }

#### 获取流程定义的最大版本

ProcessDefinitionQuery query = repositoryService.createProcessDefinitionQuery();

query.processDefinitionKey("selfhouse");

query.latestVersion();

ProcessDefinition def = query.singleResult();

defId = def.getId();

## 3.启动流程实例

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

这个方法会一直使用最新发布版本的流程定义(译者注：实际中一般都使用这种方式启动流程)。

注意，它和startProcessInstanceById方法不同。 这个方法期望使用Activiti引擎在发布时自动生成的id。， 可以通过调用processDefinition.getId()方法获得这个值。 生成的id的格式为**'key:version'**， 最大长度限制为**64个字符**， 如果你在启动时抛出了一个ActivitiException，说明生成的id太长了， 需要限制流程的key的长度。

#### 步骤：

* + 设置使用的流程定义
  + 设置启动的用户
  + 绑定业务实体的id ： businessKey  
    可以使用此业务键轻松查找流程实例
  + 为实体保存流程实例id

## 4.查找代办任务

* 直接分配给用户的任务可以通过TaskService像下面这样获取：

List<Task> tasks = taskService.createTaskQuery().taskAssignee("kermit").list();

* + 候任务也可以加入到人员的候选任务列表中

List<Task> tasks = taskService.createTaskQuery().taskCandidateUser("kermit");

* + 根据任务实体id，查找出相关联的业务key

ProcessInstance ins = runtimeService.createProcessInstanceQuery().processInstanceId (task.getProcessInstanceId()).active().singleResult();  
 String businessKey = ins.getBusinessKey();

## 5.办理任务

* + 设置办理的任务id、办理人id，设置流程变量idea（审核意见，true|false是否通过）

Map<String, Object> variable = new HashMap<>();  
 variable.put("idea", idea);

taskService.claim(taskId, userPk);  
 taskService.complete(taskId, variable);

## 6.返回追踪流程图的输入流对象

public InputStream trackProcess(String executionid) {  
 //executionid->流程实例->流程定义id->bpmn  
 ProcessInstance process = runtimeService.createProcessInstanceQuery().processInstanceId(executionid).singleResult();  
 BpmnModel bpmnmodel = repositoryService.getBpmnModel(process.getProcessDefinitionId());  
 //获得活动的节点ids  
 List<String> activeActivityIds = runtimeService.getActiveActivityIds(executionid);  
  
 // 获得历史活动记录实体（通过启动时间正序排序，不然有的线可以绘制不出来）  
 List<HistoricActivityInstance> historicActivityInstances = historyService  
 .createHistoricActivityInstanceQuery().executionId(executionid)  
 .orderByHistoricActivityInstanceStartTime().asc().list();  
 // 计算活动线  
 List<String> highLightedFlows = getHighLightedFlows(historicActivityInstances, (ProcessDefinitionEntity) ((RepositoryServiceImpl) repositoryService)  
 .getDeployedProcessDefinition(process.getProcessDefinitionId()));  
 //流程图生成器  
 DefaultProcessDiagramGenerator gen = new DefaultProcessDiagramGenerator();  
 InputStream in = gen.generateDiagram(bpmnmodel, "png", activeActivityIds,highLightedFlows,"宋体","微软雅黑","微软雅黑",null,1.0);  
  
 return in;  
}  
//获得高亮的线  
private List<String> getHighLightedFlows(List<HistoricActivityInstance> historicActivityInstances,  
 ProcessDefinitionEntity processDefinitionEntity) {  
 List<String> highFlows = new ArrayList<String>();// 用以保存高亮的线flowId  
 for (int i = 0; i < historicActivityInstances.size(); i++) {// 对历史流程节点进行遍历  
 ActivityImpl activityImpl = processDefinitionEntity  
 .findActivity(historicActivityInstances.get(i)  
 .getActivityId());// 得 到节点定义的详细信息  
 List<ActivityImpl> sameStartTimeNodes = new ArrayList<ActivityImpl>();// 用以保存后需开始时间相同的节点  
 if ((i + 1) >= historicActivityInstances.size()) {  
 break;  
 }  
 ActivityImpl sameActivityImpl1 = processDefinitionEntity  
 .findActivity(historicActivityInstances.get(i + 1)  
 .getActivityId());// 将后面第一个节点放在时间相同节点的集合里  
 sameStartTimeNodes.add(sameActivityImpl1);  
 for (int j = i + 1; j < historicActivityInstances.size() - 1; j++) {  
 HistoricActivityInstance activityImpl1 = historicActivityInstances  
 .get(j);// 后续第一个节点  
 HistoricActivityInstance activityImpl2 = historicActivityInstances  
 .get(j + 1);// 后续第二个节点  
 if (activityImpl1.getStartTime().equals(  
 activityImpl2.getStartTime())) {// 如果第一个节点和第二个节点开始时间相同保存  
 ActivityImpl sameActivityImpl2 = processDefinitionEntity  
 .findActivity(activityImpl2.getActivityId());  
 sameStartTimeNodes.add(sameActivityImpl2);  
 } else {// 有不相同跳出循环  
 break;  
 }  
 }  
 List<PvmTransition> pvmTransitions = activityImpl  
 .getOutgoingTransitions();// 取出节点的所有出去的线  
 for (PvmTransition pvmTransition : pvmTransitions) {// 对所有的线进行遍历  
 ActivityImpl pvmActivityImpl = (ActivityImpl) pvmTransition  
 .getDestination();// 如果取出的线的目标节点存在时间相同的节点里，保存该线的id，进行高亮显示  
 if (sameStartTimeNodes.contains(pvmActivityImpl)) {  
 highFlows.add(pvmTransition.getId());  
 }  
 }  
 }  
 return highFlows;  
}

## 7.查询参与的正在运行的流程实例

ProcessInstanceQuery insQuery = runtimeService.createProcessInstanceQuery();  
List<ProcessInstance> insList = insQuery.processDefinitionKey("selfhouse").involvedUser(userPK).listPage(start, limit);

## 8.查询已经结束的流程实例数据

HistoricProcessInstanceQuery query = historyService.createHistoricProcessInstanceQuery().processDefinitionKey("selfhouse").startedBy(userPK).finished();

## 9.跳转指定节点

#### 9.1 工具代码

1. **package** com.daling.ch1.jd;
2. **import** java.util.Iterator;
3. **import** java.util.Map;
4. **import** org.activiti.engine.impl.context.Context;
5. **import** org.activiti.engine.impl.interceptor.Command;
6. **import** org.activiti.engine.impl.interceptor.CommandContext;
7. **import** org.activiti.engine.impl.persistence.entity.ExecutionEntity;
8. **import** org.activiti.engine.impl.persistence.entity.ExecutionEntityManager;
9. **import** org.activiti.engine.impl.persistence.entity.TaskEntity;
10. **import** org.activiti.engine.impl.pvm.process.ActivityImpl;
12. /\*\*
13. \*
14. \* JD节点的跳转
15. \* 分享牛原创(尊重原创 转载对的时候第一行请注明，转载出处来自分享牛http://blog.csdn.net/qq\_30739519)
16. \*/
17. **public** **class** JDJumpTaskCmd **implements** Command<Void> {
18. **protected** String executionId;
19. **protected** ActivityImpl desActivity;
20. **protected** Map<String, Object> paramvar;
21. **protected** ActivityImpl currentActivity;
22. /\*\*
23. \* 分享牛原创(尊重原创 转载对的时候第一行请注明，转载出处来自分享牛http://blog.csdn.net/qq\_30739519)
24. \*/
25. **public** Void execute(CommandContext commandContext) {
26. ExecutionEntityManager executionEntityManager = Context
27. .getCommandContext().getExecutionEntityManager();
28. // 获取当前流程的executionId，因为在并发的情况下executionId是唯一的。
29. ExecutionEntity executionEntity = executionEntityManager
30. .findExecutionById(executionId);
31. executionEntity.setVariables(paramvar);
32. executionEntity.setEventSource(**this**.currentActivity);
33. executionEntity.setActivity(**this**.currentActivity);
34. // 根据executionId 获取Task
35. Iterator<TaskEntity> localIterator = Context.getCommandContext()
36. .getTaskEntityManager()
37. .findTasksByExecutionId(**this**.executionId).iterator();
39. **while** (localIterator.hasNext()) {
40. TaskEntity taskEntity = (TaskEntity) localIterator.next();
41. // 触发任务监听
42. taskEntity.fireEvent("complete");
43. // 删除任务的原因
44. Context.getCommandContext().getTaskEntityManager()
45. .deleteTask(taskEntity, "completed", **false**);
46. }
47. executionEntity.executeActivity(**this**.desActivity);
48. **return** **null**;
49. }
51. /\*\*
52. \* 构造参数 可以根据自己的业务需要添加更多的字段
53. \* 分享牛原创(尊重原创 转载对的时候第一行请注明，转载出处来自分享牛 http://blog.csdn.net/qq\_30739519)
54. \* @param executionId
55. \* @param desActivity
56. \* @param paramvar
57. \* @param currentActivity
58. \*/
59. **public** JDJumpTaskCmd(String executionId, ActivityImpl desActivity,
60. Map<String, Object> paramvar, ActivityImpl currentActivity) {
61. **this**.executionId = executionId;
62. **this**.desActivity = desActivity;
63. **this**.paramvar = paramvar;
64. **this**.currentActivity = currentActivity;
65. }
66. }

#### 9.2 使用

方式一：

1. Map<String, Object> vars = **new** HashMap<String, Object>();
2. String[] v = { "shareniu1", "shareniu2", "shareniu3", "shareniu4" };
3. vars.put("assigneeList", Arrays.asList(v));
4. //分享牛原创(尊重原创 转载对的时候第一行请注明，转载出处来自分享牛http://blog.csdn.net/qq\_30739519)
5. RepositoryService repositoryService = demo.getRepositoryService();
6. ReadOnlyProcessDefinition processDefinitionEntity = (ReadOnlyProcessDefinition) repositoryService
7. .getProcessDefinition("daling:29:137504");
8. // 目标节点
9. ActivityImpl destinationActivity = (ActivityImpl) processDefinitionEntity
10. .findActivity("usertask5");
11. String executionId = "137509";
12. // 当前节点
13. ActivityImpl currentActivity = (ActivityImpl)processDefinitionEntity
14. .findActivity("usertask3");
15. demo.getManagementService().executeCommand(
16. **new** JDJumpTaskCmd(executionId, destinationActivity, vars,
17. currentActivity));

方式二：

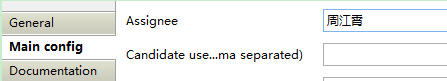
1. Map<String, Object> vars = **new** HashMap<String, Object>();
2. String[] v = { "shareniu1", "shareniu2", "shareniu3", "shareniu4" };
3. vars.put("assigneeList", Arrays.asList(v));
4. //分享牛原创(尊重原创 转载对的时候第一行请注明，转载出处来自分享牛http://blog.csdn.net/qq\_30739519)
5. CommandExecutor commandExecutor = taskServiceImpl
6. .getCommandExecutor();
7. commandExecutor.execute(**new** JDJumpTaskCmd(executionId,
8. destinationActivity, vars, currentActivity));

## 分配个人任务的三种方式

网络学习: 比较详细的分配组任务网址：https://www.2cto.com/kf/201609/549329.html

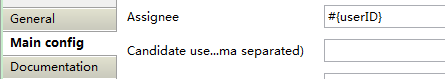
#### 9.1 分配任务

方式一：直接指定办理人

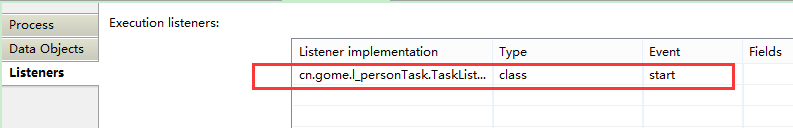
*（配置任务节点办理人）*

1. String userId = "周江霄";
2. List<Task> list = processEngine.getTaskService()//
3. .createTaskQuery()//
4. .taskAssignee(userId)//指定个人任务查询
5. .list();

方式二：直接指定办理人

*（配置任务节点变量）*

1. //启动流程实例
2. /\*\*
3. \* 启动流程实例的同时，设置流程变量，使用流程变量的方式设置下一个任务的办理人
4. \*     流程变量的名称，就是在task.bpmn中定义#{userID}的userID
5. \*     流程变量的值，就是任务的办理人
6. \*/
7. Map<String, Object> variables = **new** HashMap<String, Object>();
8. variables.put("userID", "周江霄");
9. ProcessInstance pi = processEngine.getRuntimeService()//
10. .startProcessInstanceByKey("task",variables);//使用流程定义的key的最新版本启动流程
11. System.out.println("流程实例ID："+pi.getId());
12. System.out.println("流程定义的ID："+pi.getProcessDefinitionId());
    1. //任务办理人
    2. String assignee = "周江霄";
    3. List<Task> list = processEngine.getTaskService()
    4. .createTaskQuery()
    5. .taskAssignee(assignee)//个人任务的查询
    6. .list();

方式三：使用监听器类  


*（流程图中配置任务节点监听）*

1. **package** cn.gome.l\_personTask;
3. **import** org.activiti.engine.delegate.DelegateTask;
4. **import** org.activiti.engine.delegate.TaskListener;
6. **public** **class** TaskListenerImpl **implements** TaskListener{
8. /\*\*设置任务的办理人（个人任务和组任务）\*/
9. @Override
10. **public** **void** notify(DelegateTask delegateTask) {
11. //指定个人任务
12. delegateTask.setAssignee("郭靖");
13. }
15. }

* 1. //任务办理人
  2. String assignee = "郭靖";
  3. List<Task> list = processEngine.getTaskService()//
  4. .createTaskQuery()//
  5. .taskAssignee(assignee)//个人任务的查询
  6. .list();

#### 9.2动态设置处理人

1. /\*\*将个人任务从一个人分配给另一个人\*/
2. @Test
3. **public** **void** setAssignee(){
4. //指定任务的办理人
5. String userId = "黄蓉";
6. //任务ID
7. String taskId = "5408";
8. processEngine.getTaskService()//
9. .setAssignee(taskId, userId);
10. }

## 10.监听器的使用

交给spring管理的bean，

@Service("leaveFailListener")

public class AuditLeaveFailListener implements ExecutionListener{

@Resource

private AuditLeaveDao auditLeaveDao;

@Override

public void notify(DelegateExecution execution) throws Exception {

String eventName = execution.getEventName();

if ("start".equals(eventName)) {

System.out.println("start=========");

}else if ("end".equals(eventName)) {

System.out.println("end=========");

}

else if ("take".equals(eventName)) {

System.out.println("take=========");

}

}

}