Activiti 第四天 组任务 网关

课程安排：

组任务：

Candidate-user 候选人

Candidate-group 候选组(重点)

网关(重点)：

ExclusiveGateway 排他网关

parallelGateway 并行网关

InclusiveGateway 包含网关

综合案例（重点）

# 复习

采购流程监控查询（常用，重点）

当前流程实例查询，查询系统中当前正在运行所有流程，查看流程当前运行的状态

结束的流程信息，主要应用于对历史业务数据进行统计分析：

实现方法（掌握）：

分工明确，业务系统记录了业务数据，统计来源于业务系统。

可能会需要acitiviti将流程的运行信息通过监听器写入业务系统中。

监听器：

TaskListener

ExecutionListener

历史任务查询：

根据流程实例id查询该流程执行过的所有任务

根据用户id（通常是当前用户）查询该 用户所执行过的任务

流程变量：

流程变量类型包括 ：

简单类型：string double…

序列化类型：自定义的pojo，需要实现serializable接口。

流程变量作用域：

Global全局变量：activiti中常用，作用域最大，是整个流程实例，当流程实例结束，global变量无效。

Local局部变量：作用域可以是一个任务Task、或一个执行分支Execution，当这个任务结束，local变量无效。

可以通过historyService查询历史流程变量。

注意：流程变量是activiti用于控制流程设置的变量，不建议流程变量中存储业务数据。

流程变量使用：（重点）

通过UEL表达式来使用流程变量控制流程的执行。

通过在连线上设置condition条件，条件使用UEL表达式，表达式中使用流程变量。

注意：

如果UEL表达式中流程变量不存在会跑出异常。

如果UEL表达式中流程变量存在，没有符合的条件，流程会执行结束。（排他网关可以避免）

全局变量设置：

常用：

1》流程启动时设置流程变量，流程变量可以任何结点使用。

runtimeService.startProcessInstanceByKey(processDefinitionKey,variables)

2》任务完成时设置流程变量，在任务的后续结点可以使用该流程变量。

taskService.complete(tasked, variables)

3》通过当前流程实例 的id设置流程变量

runtimeService.setVariables(processInstanceId, variables)

通过：runtimeService.getVariables（）方法获取全局变量

processInstanceId：必须是当前正在运行的流程实例id

4》通过当前待办任务的id设置流程变量。

taskService.setVariables(taskId, variables)

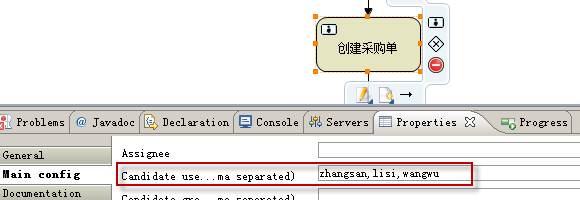
通过: taskService.getVariables()方法获取全局变量

taskId：必须是当前待办任务(未完成任务)的id

# Candidate-user候选人

## 什么是候选人

采用固定分配方法给任务指定负责人，如果任务负责人出现变更，需要修改流程定义，就可以采用候选人分配方式，先给任务分配多个候选人，候选人通过拾取组任务进行个人任务办理。



给任务分配候选人，如果分配多个候选人中间使用半角逗号分隔。

## 什么是组任务

多个候选人有资格完成该 任务，这个任务叫做组任务。

组任务具备条件：

任务没有设置assignee任务负责人

任务具有候选人

## 候选人办理任务过程

第一步：给任务设置候选人（多个, 中间使用半角逗号分隔）

候选人是无法办理任务

第二步：候选人查询组任务

使用taskService查询，指定candidate候选人。

第三步：候选人拾取(claim)组任务

**候选人拾取组任务后，该 候选人变为任务的负责人，该任务变为个人任务**

如果候选人拾取组任务后，不想办理该 任务，可以将个人任务归还，该个人任务变为组任务

第四步：查询待办个人任务

第五步：办理任务

第六步：流程结束

## Candidate-user办理任务api

候选人查询组任务

使用taskService指定candidateUser候选人查询组任务。

//任务查询对象

TaskQuery taskQuery = taskService.createTaskQuery();

//候选人

String candidateUser = "zhangsan";

taskQuery.taskCandidateUser(candidateUser);

//流程定义key

String processDefinitionKey = "purchasingflow";

taskQuery.processDefinitionKey(processDefinitionKey);

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

注意：查询组任务，必须指定 candidateUser候选人，查询该 候选人有资格办理的组任务。

拾取组任务

通过taskService，指定任务id和候选人拾取任务：

TaskService taskService = processEngine.getTaskService();

//组任务id

String taskId = "5604";

//任务候选人，claim拾取后该 候选人变为任务负责人

String userId = "zhangsan";

//任务拾取

taskService.claim(taskId, userId);

注意：如果拾取人不是该任务的候选人也可以拾取成功，在拾取之前需要校验，该 候选人是否有资格拾取该 任务.

// 组任务id

String taskId = "6004";

// 任务候选人，claim拾取后该 候选人变为任务负责人

String candidateUser = "zhangsan4";

//根据候选人和组任务id查询，如果有记录说明该 候选人有资格拾取该 任务

Task task = taskService.createTaskQuery().taskId(taskId)

.taskCandidateUser(candidateUser).singleResult();

**if**(task!=**null**){

// 任务拾取

taskService.claim(taskId, candidateUser);

System.*out*.println("任务拾取成功");

}

组任务归还

// 归还组任务，由个人任务变为组任务，还可以进行任务交接

@Test

**public** **void** setAssignee() {

// 查询任务使用TaskService

TaskService taskService = processEngine.getTaskService();

// 当前待办任务

String taskId = "6004";

// 任务负责人

String userId = "zhangsan2";

//校验userId是否是taskId的负责人，如果是负责人才可以归还组任务

Task task = taskService.createTaskQuery().taskId(taskId).taskAssignee(userId).singleResult();

**if**(task!=**null**){

//如果设置为null，归还组任务,该 任务没有负责人

taskService.setAssignee(taskId, **null**);

}

}

任务交接

任务负责人也可以将任务交给其它候选人办理该任务

代码如下：

@Test

**public** **void** setAssigneeToCandidateUser() {

// 查询任务使用TaskService

TaskService taskService = processEngine.getTaskService();

// 当前待办任务

String taskId = "6004";

// 任务负责人

String userId = "zhangsan2";

// 校验userId是否是taskId的负责人，如果是负责人才可以归还组任务

Task task = taskService.createTaskQuery().taskId(taskId)

.taskAssignee(userId).singleResult();

**if** (task != **null**) {

// 将此任务交给其它候选人办理该 任务

String candidateuser = "zhangsan";

// 根据候选人和组任务id查询，如果有记录说明该 候选人有资格拾取该 任务

Task task2 = taskService.createTaskQuery().taskId(taskId)

.taskCandidateUser(candidateuser).singleResult();

**if** (task2 != **null**) {

// 才可以交接

taskService.setAssignee(taskId, candidateuser);

}

}

}

查询个人任务

参考个人任务章节

办理个人任务

参考个人任务章节

数据表跟踪

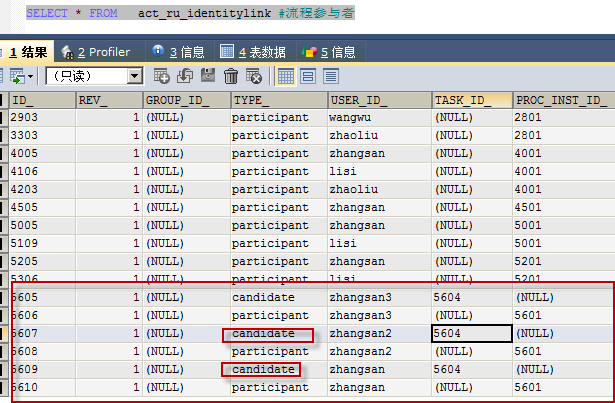
如果任务设置候选人，当前任务表中assignee（任务负责人）是空。

SELECT \* FROM act\_ru\_task #当前任务表



SELECT \* FROM act\_ru\_identitylink #流程参与者

如果任务设置候选，记录所有候选人信息



任务拾取后，task表中assignee记录任务的负责人



# Candidate-group候选组

## 什么候选组

即使给任务指定了多个候选人，多个候选人都有办理任务资格，但是候选的人数有限，无法动态扩展，如果需要添加或删除候选，需要修改流程定义 文件，不利于系统 扩展。

采用候选组方式解决上边的问题。

给任务设置候选组，在组中有多个用户并且可以动态扩展用户，组中的用户都是候选人，候选人先拾取组任务，将组任务变为自己的个人任务，进行个人任务办理。

## 候选组办理任务过程

第一步：Activiti会自动从候选组中找用户，将这些用户作为该 任务的候选人。

下边的流程同候选人办理任务过程！！

第二步：给任务设置候选人（多个, 中间使用半角逗号分隔）

候选人是无法办理任务

第三步：候选人查询组任务

使用taskService查询，指定candidate候选人。

第四步：候选人拾取(claim)组任务

**候选人拾取组任务后，该 候选人变为任务的负责人，该任务变为个人任务**

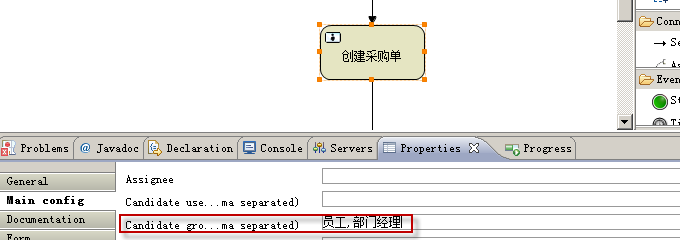
如果候选人拾取组任务后，不想办理该 任务，可以将个人任务归还，该个人任务变为组任务

第五步：查询待办个人任务

第六步：办理任务

第七步：流程结束

## 设置候选组



多个组中间使用半角逗号分隔。

## 设置组和用户信息

Activiti中采用以下表记录组信息、用户信息、组和用户关系 信息

SELECT \* FROM act\_id\_group #组信息

SELECT \* FROM act\_id\_user #用户信息

SELECT \* FROM act\_id\_membership #组和用户关系信息

Api设置方法

以下设置的信息和业务系统的用户信息、角色信息保存一致。

先设置组信息



再设置用户信息



再设置组和用户关系信息

代码如下：

//设置组和用户信息

@Test

**public** **void** setUserGroup(){

IdentityService identityService = processEngine.getIdentityService();

//设置组信息

//添加之前应该校验组信息是否存在，不存在再进行添加

**if**(identityService.createGroupQuery().groupId("10").singleResult()==**null**){

//添加新组

GroupEntity groupEntity = **new** GroupEntity();

groupEntity.setId("10");

groupEntity.setName("员工");

identityService.saveGroup(groupEntity);

}

**if**(identityService.createGroupQuery().groupId("11").singleResult()==**null**){

//添加新组

GroupEntity groupEntity = **new** GroupEntity();

groupEntity.setId("11");

groupEntity.setName("部门经理");

identityService.saveGroup(groupEntity);

}

**if**(identityService.createGroupQuery().groupId("12").singleResult()==**null**){

//添加新组

GroupEntity groupEntity = **new** GroupEntity();

groupEntity.setId("12");

groupEntity.setName("总经理");

identityService.saveGroup(groupEntity);

}

**if**(identityService.createGroupQuery().groupId("13").singleResult()==**null**){

//添加新组

GroupEntity groupEntity = **new** GroupEntity();

groupEntity.setId("13");

groupEntity.setName("财务");

identityService.saveGroup(groupEntity);

}

//设置用户信息

//添加之前应该校验用户信息是否存在，不存在再进行添加

**if**(identityService.createUserQuery().userId("zhangsan").singleResult()==**null**){

//添加新用户

UserEntity userEntity = **new** UserEntity();

userEntity.setId("zhangsan");

userEntity.setFirstName("张三");

identityService.saveUser(userEntity);

}

**if**(identityService.createUserQuery().userId("lisi").singleResult()==**null**){

//添加新用户

UserEntity userEntity = **new** UserEntity();

userEntity.setId("lisi");

userEntity.setFirstName("李四");

identityService.saveUser(userEntity);

}

**if**(identityService.createUserQuery().userId("wangwu").singleResult()==**null**){

//添加新用户

UserEntity userEntity = **new** UserEntity();

userEntity.setId("wangwu");

userEntity.setFirstName("王五");

identityService.saveUser(userEntity);

}

**if**(identityService.createUserQuery().userId("zhaoliu").singleResult()==**null**){

//添加新用户

UserEntity userEntity = **new** UserEntity();

userEntity.setId("zhaoliu");

userEntity.setFirstName("赵六");

identityService.saveUser(userEntity);

}

//设置用户和组的关系信息

//采用先删除再添加

identityService.deleteMembership("zhangsan", "10");

identityService.createMembership("zhangsan", "10");

identityService.deleteMembership("lisi", "11");

identityService.createMembership("lisi", "11");

identityService.deleteMembership("wangwu", "12");

identityService.createMembership("wangwu", "12");

identityService.deleteMembership("zhaoliu", "13");

identityService.createMembership("zhaoliu", "13");

}

与业务系统 同步方法(常用)

**业务系统 中存在用户信息，activiti中也存在用户信息，需要只维护一处数据。**

**通常采用业务系统 向activiti同步方法：**

需要将业务系统 中的用户信息和角色信息同步到activiti中。

数据同步：

**业务系统---》activiti同步**

方法1 :

数据库触发器方法

企业中在进行数据同步的常用方法，一般在一个数据库中采用此方法。

业务系统 用户表----》activiti的act\_id\_user

在业务系统 用户表添加触发器：新增、删除、修改

注意：如果act\_id\_user有外键关系，需要先删除依赖关系。

业务系统 角色表----》activiti的act\_id\_group

业务系统 角色和用户关系表—》activiti的act\_id\_membership

方法2 :

采用即时触发java程序。

* 用户角色同步：

在操作业务系统 用户角色表时执行以下操作：

业务系统添加角色-🡪activiti添加角色

业务系统修改角色🡪activiti修改角色

业务系统删除角色🡪activiti删除角色，删除之前将用户角色关系表先删除（根据角色删除）

* 用户信息同步

在操作业务系统 用户表时执行以下操作

业务系统添加用户----》activiti添加用户，添加用户与角色关系表

业务系统修改用户---》activiti修改用户，先删除原来用户与角色关系表，再添加用户与角色关系

业务系统删除用户—》activiti删除用户，删除之前将用户角色关系表删除（根据用户删除）

## 组任务办理过程api

设置组和用户信息

参考上边设置组和用户api

正式开发时，需要将业务系统 用户和角色信息同步到activiti中。

候选人查询组任务

参考candidate-user的api

注意：

在activiti的用户、组、用户和组关系表中随时添加数据，不受流程启动先后顺序影响。

拾取组任务

参考candidate-user的api

查询个人任务

参考candidate-user的api

办理个人任务

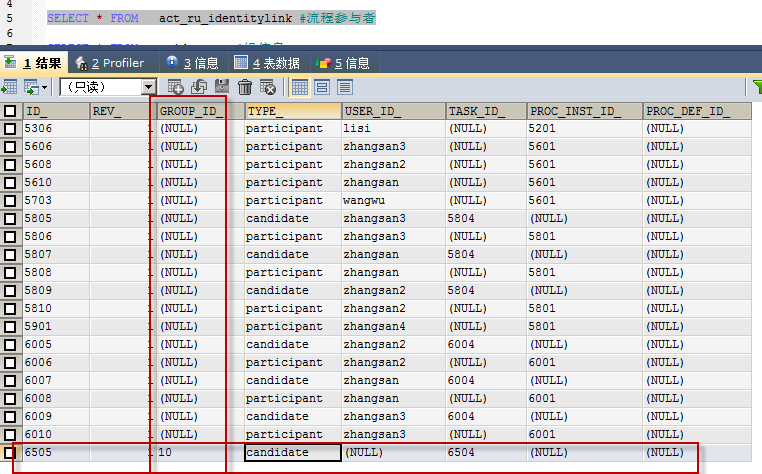
参考candidate-user的api

数据表跟踪

和候选人不一样：

SELECT \* FROM act\_ru\_identitylink #流程参与者

在group\_id\_字段设置了候选组的id



# 网关

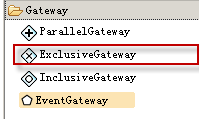
## 排他网关

什么排他网关

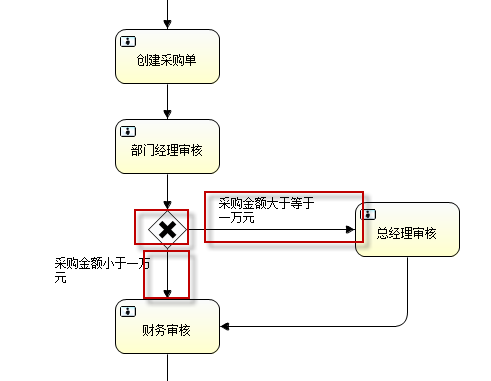
排他网关用于决策，选择分支执行流程，分支上需要设置condition条件，如果分支的条件结果 为true，那么该 分支会通过排他网关。排他网关只会选择一条分支去执行。

定义方法

图标：



流程定义：



排他网关测试

第一步：流程定义部署

第二步：启动流程实例

设置price流程变量值，因为price 在排他网关的两分支使用

第三步：查询待办任务

也可以在部门经理审核后设置price流程变量值，因为price 在排他网关的两分支使用

第四步：办理任务

如果分支上的条件都不满足，没有一条线经过排他网关，activiti会抛出异常：

org.activiti.engine.ActivitiException: No outgoing sequence flow of the exclusive gateway 'exclusivegateway1' could be selected for continuing the process

at org.activiti.engine.impl.bpmn.behavior.ExclusiveGatewayActivityBehavior.leave(ExclusiveGatewayActivityBehavior.java:85)

如果多条分支都 满足，只会有一条线经过排他网关。

上边两种情况必须在开发避免！！！

## 并行网关

什么并行网关

并行网关（parallelGateway），包括分支和汇聚两个结点，所有的分支不判断条件都经过分支结点，所有经过分支结点的分支都要进行汇聚，所有的分支全部执行完成，并行网关执行完成。

Fork(分支)

所有的分支不判断条件都经过分支结点

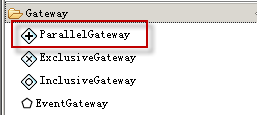
Join（汇聚）

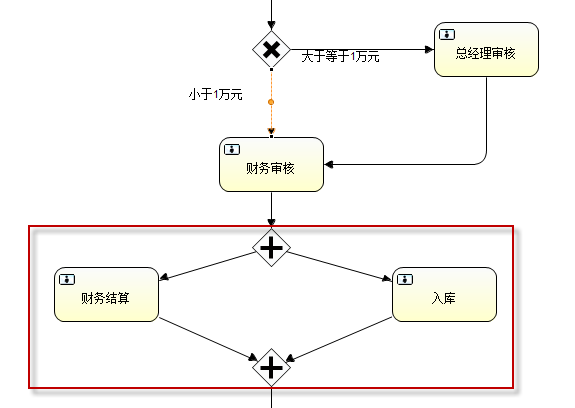
所有经过分支结点的分支都要进行汇聚

分支的数量等于汇聚数量!

流程定义

图标：



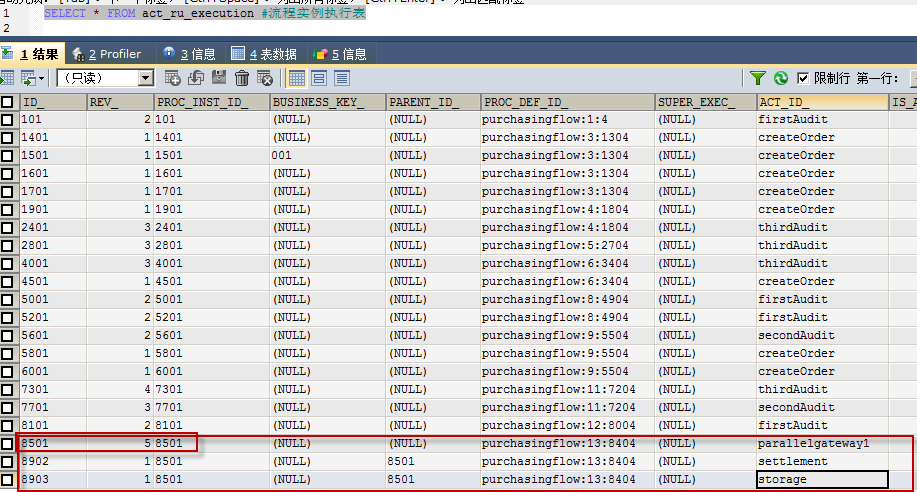


注意：经过并行网关的分支结点，不需要设置condition条件。

并行网关测试

当流程执行到并行的分支结点时，

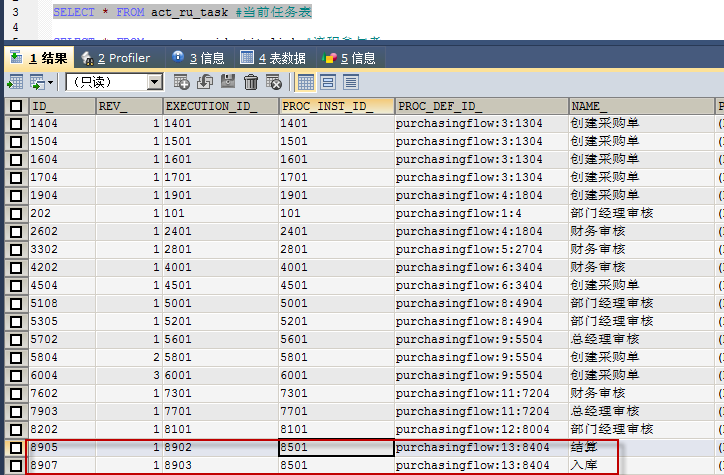
向act\_ru\_execution #流程实例执行表执行并行分支（结算，入库）



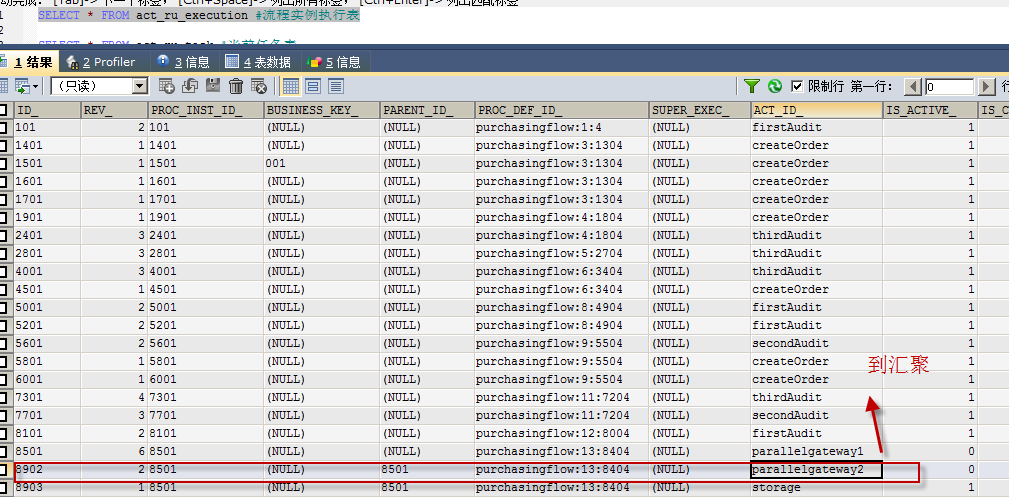
Execution表中8501的记录数等于分支数+1

只有一条记录的流程实例 id和流程实例执行id相等的，这一条为流程执行主线。

向当前任务表中插入两条记录（结算、入库）：



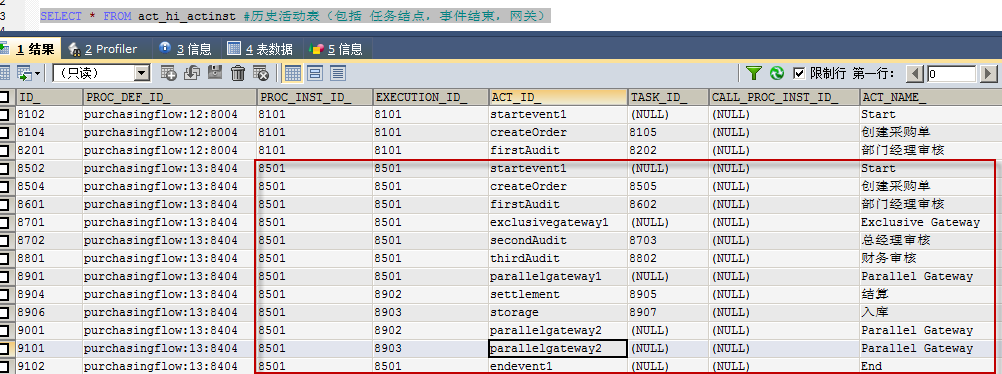
如果一条分支执行完成，在execution表中act\_id\_改为并行网关汇聚结点的id，表示分支执行完成到汇聚结点等待其它分支。



当一个分支结束，在当前任务表中就删除该 分支的任务。

所有的分支执行完成，到汇聚结点，并行网关执行完成。

所有经过的结点如下：



历史活动表中，有几条分支就有几条汇聚。

## 包含网关

什么是包含网关

包含网关是排他网关和并行网关的结合体。

包含网关（IncluesiveGateway），包括分支和汇聚两个结点，经过分支结点需要判断条件，满足条件经过分支结点，所有经过分支结点的线边最终会进行汇聚。

Fork(分支)

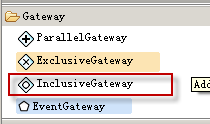
所有的分支需要判断条件，满足条件的经过分支结点

Join(汇聚)

所有满足条件的分支都要进行汇聚

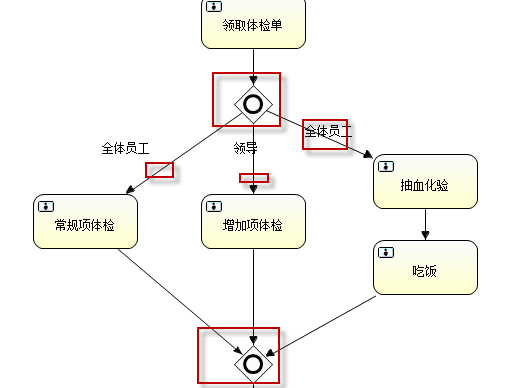
流程定义

图标:



员工类型：

通过流程变量userType来表示，如果等于1表示普通员工，如果等于2表示领导



注意：通过包含网关的每个分支的连线上设置condition条件。

需求如下：

领取完成体检单，对于普通员工体检内容包括 （常规项、抽血化验），对于领导体检内容包括 （常规项、抽血化验、增加项体检），

其中，抽血化验完成方可吃饭，吃饭完成表示抽血化验分支就完成。

设置candition条件：

常规项体检：${userType==’1’ || userType=’2’}

增加项体检：${userType==’2’}

抽血化验：${userType==’1’ || userType=’2’}

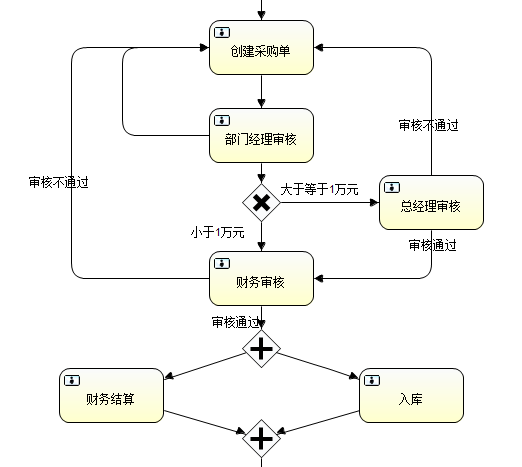
包含网关测试

包含网关与并行网关的不同就是经过分支结点的需要满足条件才进行汇聚，并行网关不判断条件，所有经过分支的都经过汇聚结点。

# 案例

## 需求

* 将采购流程改为组任务(使用候选组)实现
* 在采购流程中实现排他网关
* 在采购流程中实现并行网关



需求描述：

员工创建采购单

经过部门经理审核

审核通过：

部门经理审核通过，如果采购金额大于等于1万元，由总经理审核

部门经理审核通过，如果采购金额小于1万元，由财务审核

审核不通过：

部门经理审核不通过，由员工重新修改采购单进行提交

总经理审核

总经理审核通过由财务审核通过

总经理审核不通过，由员工重新修改采购单进行提交

财务审核

财务审核通过并行执行财务结算和入库

财务审核不通过，由员工重新修改采购单进行提交

财务结算和入库两个操作可以并行执行。

## 流程定义

部门经理审核通过后，通过排他网关决定走总经理审核还是财务审核。

财务审核通过后，经过并行网关，财务结算和入库并行执行。

审核分支设置condition 条件：

部门经理审核 ：

审核通过candition：

${order.price>=10000 && firstAudit. status==’1’} 部门经理审核通过且采购金额大于等于10000元由总经理审核

${order.price<10000 && firstAudit. status==’1’}部门经理审核通过且采购金额小于10000元由财务审核

firstAudit和order都是流程变量名称

审核 不通过candition :

${ firstAudit. status==’0’}：部门经理审核不通过由流程发起人重新修改采购单再提交

总经理审核 ：

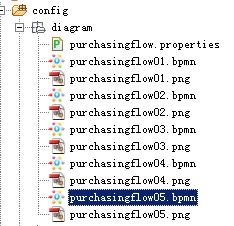
审核通过candition：${ secondAudit. status==’1’} 总经理审核通过

审核 不通过candition : ${ secondAudit. status==’0’}总经理审核不通过

财务审核 ：

审核通过candition：${ thirdAudit. status==’1’} 财务审核通过

审核 不通过candition : ${ third Audit. status==’0’}财务审核不通过



## 分析

需要开发结算和入库功能

真正的结算和入库很复杂，确定一个功能将流程向后推进一步。

为了教学方便，开发结算和入库功能实现activiti任务完成。

候选人查询组任务

由于任务结点改变 候选组方式分配任务，实现候选人查询组任务。

实现 方法：

调用taskService，指定candidateUser候选人查询组任务。

候选人拾取组任务

由于任务结点改变 候选组方式分配任务，实现候选拾取组任务。

实现方法：

调用taskService，指定组任务id和candidateUserId候选人。

拾取任务之前需要校验候选人是否资格拾取该组任务。

## 开发

结算和入库功能

dao

不用开发。

实际采购系统 需要开发的，执行结算和入库。

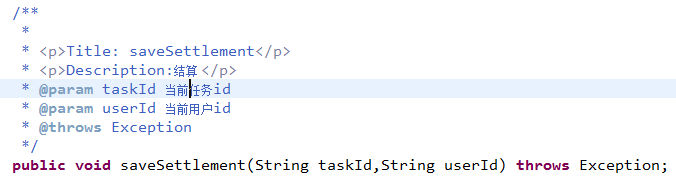
service

接口功能：结算

接口参数：taskId任务id、userId用户id

接口实现：

调用 activiti的api执行任务完成

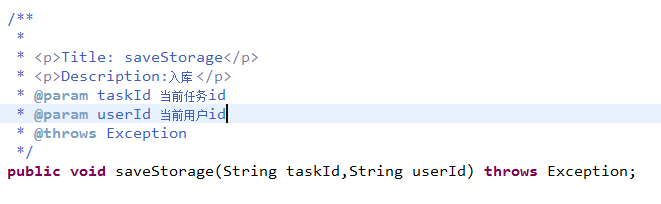


接口功能：入库

接口参数：taskId任务id、userId用户id

接口实现：

调用 activiti的api执行任务完成



@Override

**public** **void** saveSettlement(String taskId, String userId) **throws** Exception {

// 根据任务id和assignee查询该任务

Task task = taskService.createTaskQuery().taskId(taskId)

.taskAssignee(userId).singleResult();

**if** (task != **null**) {

taskService.complete(taskId);

}

}

@Override

**public** **void** saveStorage(String taskId, String userId) **throws** Exception {

// 根据任务id和assignee查询该任务

Task task = taskService.createTaskQuery().taskId(taskId)

.taskAssignee(userId).singleResult();

**if** (task != **null**) {

taskService.complete(taskId);

}

}

action

两个方法：

结算方法：

接收页面传入的个人任务id参数

入库方法：

接收页面传入的个人任务id参数

// 结算

@RequestMapping("/settlement")

**public** String settlement(HttpSession session, String taskId)

**throws** Exception {

// 当前登陆用户

ActiveUser activeuser = UserUtil.*getUserFromSession*(session);

// 用户id

String userId = activeuser.getUserid();

orderService.saveSettlement(taskId, userId);

// 返回到采购单处理列表

**return** "redirect:orderTaskList.action";

}

// 入库

@RequestMapping("/storage")

**public** String storage(HttpSession session, String taskId) **throws** Exception {

// 当前登陆用户

ActiveUser activeuser = UserUtil.*getUserFromSession*(session);

// 用户id

String userId = activeuser.getUserid();

orderService.saveStorage(taskId, userId);

// 返回到采购单处理列表

**return** "redirect:orderTaskList.action";

}

页面

修改采购单处理列表页面，添加“结算”和“入库”连接：

<c:if test=*"*${order.taskDefinitionKey=='settlement' }*"*>

<a

href=*"*${baseurl}*/orderflow/settlement.action?taskId=*${order.taskId}*"*>财务结算</a>

</c:if>

<c:if test=*"*${order.taskDefinitionKey=='storage' }*"*>

<a

href=*"*${baseurl}*/orderflow/storage.action?taskId=*${order.taskId}*"*>入库</a>

</c:if>

候选人查询组任务

dao

根据 采购单id查询采购表记录。

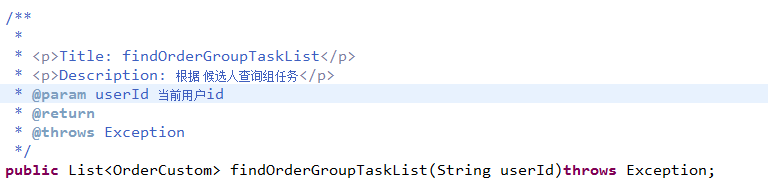
service

接口功能：候选人查询组任务

接口参数：候选人candidateUserId，（实际开发需要查询条件）

接口实现 ：

调用taskService，指定candidateUser候选人查询组任务



**public** List<OrderCustom> findOrderGroupTaskList(String userId)

**throws** Exception {

// 任务查询对象

TaskQuery taskQuery = taskService.createTaskQuery();

// 候选人，在act\_id\_user表中存在，从act\_id\_membership通过group\_id\_查询出用户

String candidateUser = userId;

// 指定候选人

taskQuery.taskCandidateUser(candidateUser);

// 流程定义key

String processDefinitionKey = ResourcesUtil.*getValue*(

"diagram.purchasingflow", "purchasingProcessDefinitionKey");

taskQuery.processDefinitionKey(processDefinitionKey);

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

List<OrderCustom> orderList = **new** ArrayList<OrderCustom>();

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

OrderCustom orderCustom = **new** OrderCustom();

//下边的代码同采购单处理列表代码...

// 流程实例id

String processInstanceId = task.getProcessInstanceId();

// 根据流程实例id找到流程实例对象

ProcessInstance processInstance = runtimeService

.createProcessInstanceQuery()

.processInstanceId(processInstanceId).singleResult();

// 从流程实例对象中获取businessKey

String businessKey = processInstance.getBusinessKey();

// 根据businessKey查询业务系统

// 采购单id

String orderId = businessKey;

PurBusOrder purBusOrder = purBusOrderMapper

.selectByPrimaryKey(orderId);

// 获取采购单名称、采购金额等采购单信息

// 将purBusOrder内容拷贝到orderCustom

BeanUtils.*copyProperties*(purBusOrder, orderCustom);

// 下边向orderCustom开始设置任务信息

// 任务id、任务标识 、任务名称

// 任务id

orderCustom.setTaskId(task.getId());

// 任务标识

orderCustom.setTaskDefinitionKey(task.getTaskDefinitionKey());

// 任务名称

orderCustom.setTaskName(task.getName());

orderList.add(orderCustom);

}

**return** orderList;

}

action

查询组任务方法：

代码基本上同采购单处理列表的代码：

// 采购单组任务列表

@RequestMapping("/orderGroupTaskList")

**public** String orderGroupTaskList(HttpSession session, Model model)

**throws** Exception {

// 当前登陆用户

ActiveUser activeuser = UserUtil.*getUserFromSession*(session);

// 用户id

String userId = activeuser.getUserid();

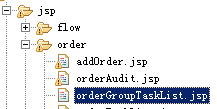
List<OrderCustom> list = orderService.findOrderGroupTaskList(userId);

model.addAttribute("list", list);

**return** "order/orderGroupTaskList";

}

页面



候选人拾取组任务

dao

不用开发。

service

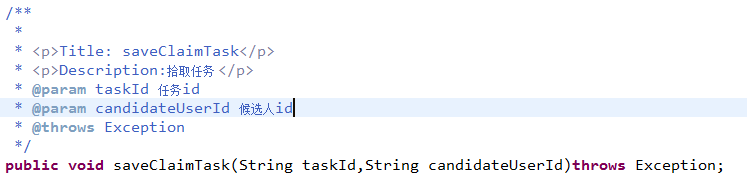
接口功能：拾取组任务

接口参数：taskId任务id，candidateUserId候选人

接口实现 ：

调用taskService，指定组任务id和candidateUserId候选人。

拾取任务之前需要校验候选人是否资格拾取该组任务。



@Override

**public** **void** saveClaimTask(String taskId, String candidateUserId)

**throws** Exception {

// 根据候选人和组任务id查询，如果有记录说明该 候选人有资格拾取该 任务

Task task = taskService.createTaskQuery().taskId(taskId)

.taskCandidateUser(candidateUserId).singleResult();

**if** (task != **null**) {

// 任务拾取

taskService.claim(taskId, candidateUserId);

System.*out*.println("任务拾取成功");

}

}

action

拾取组任务方法：

需要从页面传入taskId组任务id.

// 拾取组任务

@RequestMapping("/claimTask")

**public** String claimTask(HttpSession session, String taskId)

**throws** Exception {

// 当前登陆用户

ActiveUser activeuser = UserUtil.*getUserFromSession*(session);

// 用户id

String userId = activeuser.getUserid();

orderService.saveClaimTask(taskId, userId);

//返回采购单组任务列表

**return** "redirect:orderGroupTaskList.action";

}

页面

修改组任务列表页面，添加“拾取组任务”连接：



## 测试

测试注意点：

1、组任务(使用候选组)

准备组和用户的数据，调用activiti的api设置组、用户、组和用户关系信息

正式开发，将业务系统 用户角色数据实时同步到activiti中。

2、测试组任务办理流程

a>查询组任务功能

b>拾取组任务功能

3、测试排他网关

修改原来的功能是否存在bug。

4、测试并行网关

结算功能和入库功能是否并行执行.

# 课程总结

什么是工作流？

工作流是通过计算机自动管理业务流程，实现多个参与者按照预定义的流程自动执行业务流程。

什么是activiti？

Activiti是一个工作流的引擎（框架，jar、组件），对业务流程的自动化管理。Activiti按照bpmn2.0标准进行流程定义，按照定义流程(bpmn文件) 去自动执行业务流程。

第一步：线下进行流程定义

Activiti按照bpmn2.0标准进行流程定义，定义文件包括 .bpmn和.png，其中.bpmn是必须的文件。

第二步：进行流程定义部署

将线下流程定义文件.bpmn部署到activiti的数据库中。这样activiti方可按照定义流程(bpmn文件) 去自动执行业务流程。

两种方法部署方法：

1. 单个文件部署方法
2. Zip包部署方法

建议使用单个文件 部署，建议一次部署只部署一个流程定义 。

第三步：启动一个流程 实例

流程定义和流程实例 的区别：

流程定义 是一个静态文件(.bpmn)，流程实例是动态的流程执行过程。如果要activiti去管理业务流程的执行，需要首先发起一个流程实例 。

第四步：查询待办任务

个人任务：

通过固定设置方法或UEL表达式设置方法设置task结点的assignee属性。

组任务：

通过设置多个候选人或多个候选组的方法。

组任务常用，组任务中以candidate-group候选组方式最常用。

组任务办理过程：

通过候选人查询任务

拾取任务

同个人任务办理流程。

第五步：办理任务

流程的执行可能需要通过流程变量决定执行分支。

流程变量：

常用全局变量

需要在启动流程实例 或完成任务时设置流程变量

在UEL表达式中使用流程变量

网关：

排他网关：

经过排他网关的分支只有一条。

并行网关：

经过并行网关分支结点所有分支不管条件是否满足都 会经过并行网关的汇聚结点。

并行网关常用于会签任务（多个用户共同办理的任务）

包含网关：

经过包含网关分支结点，只有满足条件的分支才经过包含网关的汇聚结点。

第六步：流程完成

所有结点完成执行完成，流程自动完成。

注意：如果使用并行网关或包含网关，必须经过分支的流程全部执行完成才能到达汇聚结点，并行网关或包含风关就结束。