Activiti第三天 采购流程监控 流程变量 连线分支

第三天：

采购流程监控：（重点）

进行**采购业务流程**的监控：

查询当前正在运行的流程

实现动态图（在流程定义图上标出当前结点的位置，使用红色的框）

查询结束的流程

查询某个流程下历史任务（从流程开始运行到当前所经历的所有任务）

查询某个用户所办理的历史任务

流程变量：（重点）

Global全局变量（掌握）

Local局部变量（了解）

连接分支：

需求流程变量支持

设置连接的candition条件实现分支

案例1（重点）

案例2（重点）

# 复习

什么是流程定义 ？

流程定义是按照bpmn2.0标准定义业务流程，将流程定义的文件（.bpmn和.png(不是必须的)）部署到activiti中，activiti就可以管理该业务流程。

什么是流程实例 ？

参与者（可以用户，也可以程序）按照流程定义发起一个流程，这个流程就是一个流程实例 。

流程定义的内容就是一个**静态**文件(.bpmn)，流程实例的内容是该 流程的执行过程(**动态**)。

如何启动一个流程实例 ？

常用方式：启动一个流程实例时指定一个业务标识。

// 业务标识 ，如果 是采购流程就是采购单id

String businessKey = "001";

// 启动流程实例时指定业务标识

String processDefinitionKey = "purchasingflow";

ProcessInstance processInstance = runtimeService

.startProcessInstanceByKey(processDefinitionKey, businessKey);

businessKey:业务标识，作用：通过activiti的api查询activiti的流程数据时，可以通过businessKey关联查询业务系统 的数据，通常业务标识记录业务系统 表的主键，比如：如果采购流程，businesskey就是采购单id，如果是请假流程，businessKey就是请假单(请假信息表)的id。

启动一个流程实例 后，该 流程运行到第一个结点，activiti需要给该 任务结点分配任务负责人。

任务分配三种方式：

第一种：采用固定分配方法，设置task结点的assignee（任务负责人）属性，不常用。

第二种：采用UEL表达式，表达使用流程变量设置任务负责人，在企业开发中常用。

第三种：采用监听器（自定义监听器实现 TaskListener接口），可以在监听器中扩展代码，在企业开发中常用。

查询待办任务：

查询待办任务中如果包括 业务系统 数据，如果查询？

通过Taskid得到任务所属的流程实例id

通过流程实例id得到流程实例对象ProcessInstance。

从ProcessInstance获取businessKey

通过businessKey关联查询业务数据

办理任务（完成任务）：

需要参数：任务id（activiti的api要求），用户id（进行权限校验使用）

在完成任务之前需要校验该 **用户**是否有该 **任务**的完成权限。

Activiti开发步骤:

1. 在需求阶段，分析出业务流程
2. 设计阶段，确定哪些业务流程由activiti管理，对工作流管理的流程进行流程定义

流程定义时和功能设计同步进行：

1. 确定流程启动所对应的功能
2. 确定流程执行中哪些功能和流程结点对应，哪个功能可以将流程向后推进一步

Activiti开发遵循原则：

1. 角色分工明确，activiti负责流程管理 ，业务系统 负责业务功能。
2. 业务系统 中通常在service层将activiti和控制层、持久层进行隔离（解耦），比如在业务功能中需要查询activiti的流程数据，需要自定义一个对象存储activiti的数据。
3. 数据共享问题，在activiti中存储businesskey（业务标识），通过businesskey查询业务系统 数据，在业务系统 中存储activiti的标识（比如在采购单中存储流程实例 的id），在查询业务数据时通过此流程实例 id查询activiti的数据。达到目标：activiti和业务系统 能互相关联查询。

# 查询当前正在运行的流程

## 需求

查询系统 中没有结束的采购流程有哪些，查询当前运行到哪个结点，及所完成的任务列表。

查询内容：流程实例 id、当前结点、采购单名称、采购金额、下单人。

## 实现

分析:

流程实例 id、当前结点：

从activiti的数据库（act\_ru\_execution流程实例执行表）查询。

通过RuntimeService查询当前运行的流程实例 。

采购单名称、采购金额、下单人：

从业务系统数据库（pur\_bus\_order采购单表）查询。

通过采购单id查询，采购单id就是activiti流程实例表中存储的businessKey。

dao

通过采购单id查询采购单表的记录：

使用逆向工程 生成的 mapper

service

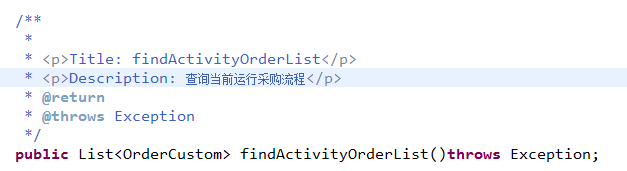
接口功能：查询当前正在运行的流程

接口参数：查询条件，如果此接口可以查询所有业务流程实例，查询条件必须将流程定义的key(可能包括：采购流程、请假流程。。)传入

接口实现：

通过RuntimeService查询当前运行的流程实例

得到businessKey关联查询业务系统 数据



@Override

**public** List<OrderCustom> findActivityOrderList() **throws** Exception {

// 创建查询对象

ProcessInstanceQuery processInstanceQuery = runtimeService

.createProcessInstanceQuery();

// 设置查询条件

//流程定义 key

String processDefinitionKey = ResourcesUtil.*getValue*(

"diagram.purchasingflow", "purchasingProcessDefinitionKey");

// 指定 流程定义key只查询该类流程的实例，比如key为采购流程，只查询采购流程实例

processInstanceQuery.processDefinitionKey(processDefinitionKey);

//当前排序字段

processInstanceQuery.orderByProcessInstanceId().desc();

// 获取查询列表

List<ProcessInstance> list = processInstanceQuery.list();

// 单独 定义一个list，list中包括自定义的pojo(OrderCustom包括 流程实例 信息和业务系统 信息)

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

**for** (ProcessInstance processInstance : list) {

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

// 比如key为采购流程，这个key就是采购单id

String businesskey = processInstance.getBusinessKey();

// 根据 businessKey获取采购单信息

PurBusOrder purBusOrder = purBusOrderMapper.selectByPrimaryKey(businesskey);

//将采购单信息拷贝到orderCustom中

BeanUtils.*copyProperties*(purBusOrder, orderCustom);

//向orderCustom中填充流程实例 信息

//当前运行的结点

orderCustom.setActivityId(processInstance.getActivityId());

orderList.add(orderCustom);

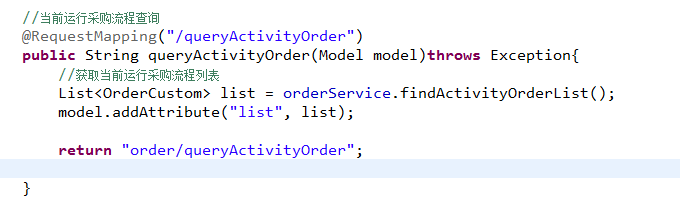
}

**return** orderList;

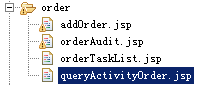
}

action

当前运行流程查询方法：



页面



测试

测试步骤：

查询当前运行的流程，对照act\_ru\_execution表。

员工创建一个新的采购单，查询当前运行的流程应该包括 该 流程。

完成一个流程，查询当前运行的流程是查询不到。

如果有查询条件，对每个查询条件进行测试。

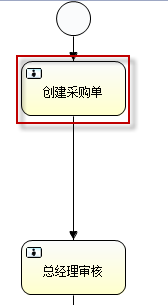
# 当前流程动态图

## 需求

通过图形展示出当前流程中当前结点位置。

效果：

在当前结点用红框标识：



## 分析

实现1：根据流程实例 id获取图形

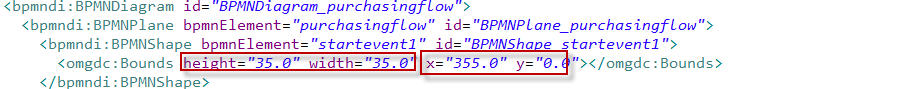
通过/flow/queryProcessDefinitionResource.action？processDefinitionId=XXXXXX& resourceType=png 这个url展示一个业务流程图。

processDefinitionId如何获取：

根据 流程实例的id查询出流程实例 的对象，从对象 中获取processDefinitionId。

实现 2：获取当前结点的坐标及宽高，根据 坐标及宽高使用红色的div层在html显示。

从流程定义文件中获取：



代码如下：

// 根据 流程定义 id查询流程定义 实体对象

ProcessDefinitionEntity processDefinitionEntity = (ProcessDefinitionEntity) repositoryService

.getProcessDefinition(processDefinitionId);

//从流程定义 实体对象查询结点的坐标和宽高

ActivityImpl activityImpl = processDefinitionEntity.findActivity(activityId);

**int** activity\_x= activityImpl.getX();//坐标

**int** activity\_y = activityImpl.getY();//坐标

**int** activity\_width = activityImpl.getWidth();//宽

**int** activity\_height = activityImpl.getHeight();//高

最终实现 ：页面上显示图形，再显示div层，图形结点的坐标及宽高与div层一致

## 图形在页面显示

action

在FlowAction添加方法：

根据流程实例 id展示动态图形：

//当前运行流程中当前结点图形

@RequestMapping("/queryActivityMap")

**public** String queryActivityMap(Model model,String processInstanceId)**throws** Exception{

//根据 流程实例的id查询出流程实例 的对象，从对象 中获取processDefinitionId。

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

String processDefinitionId=processInstance.getProcessDefinitionId();

// 将流程定义 id传到页面，用于图形显示

model.addAttribute("processDefinitionId", processDefinitionId);

//

**return** "flow/queryActivityMap";

}

页面

定义显示动态图形页面：

定义 img，src=/flow/queryProcessDefinitionResource.action？processDefinitionId=${processDefinitionId}& resourceType=png

<!-- 流程图 -->

<img style="position: *absolute*; top: *0px*; left: *0px*;"

src=*"*${baseurl}*flow/queryProcessDefinitionResource.action?processDefinitionId=*${processDefinitionId}*&resourceType=png"*>

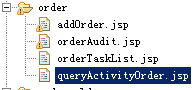
根据action传来的坐标和宽高，定义 div层：

<!-- 流程图中当前活动框 -->

<div

style=*"position: absolute;border:1px solid red;width:* ${activity\_width }*px;height:*${activity\_height }*px;top:*${activity\_y }*px;left:* ${activity\_x }*px;"*></div>

在当前运行流程页面，添加查询动态图形连接：



<td class=*category*><a

href=*"*${baseurl }*flow/queryActivityMap.action?processInstanceId=*${order.processinstanceId}*"*

target=*"\_blank"*>查看流程图</a></td>

# 查询结束的流程

## 需求

需求1、查询已结束 的采购流程。

查询内容：流程实例 id、执行开始时间、执行结束时间、采购单名称、采购金额、下单人

需求2、对已结束的采购流程进行统计，比如按分类统计采购金额总数。

## 实现查询已结束 的采购流程

api

//查询历史 流程实例

@Test

**public** **void** queryHistoryProcessInstance(){

HistoryService historyService = processEngine.getHistoryService();

//创建历史流程实例 查询对象

HistoricProcessInstanceQuery historicProcessInstanceQuery = historyService.createHistoricProcessInstanceQuery();

//设置查询条件

//指定流程定义key，只查询某个业务流程的实例

String processDefinitionKey= "purchasingflow";

historicProcessInstanceQuery.processDefinitionKey(processDefinitionKey);

//设置只查询已完成的流程实例

historicProcessInstanceQuery.finished();

//数据列表

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

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

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

System.*out*.println("流程实例所属流程定义id :"

+ historicProcessInstance.getProcessDefinitionId());

System.*out*.println("流程实例的id：" + historicProcessInstance.getId());

System.*out*.println("业务标识 ：" + historicProcessInstance.getBusinessKey());

System.*out*.println("开始执行时间："+historicProcessInstance.getStartTime());

System.*out*.println("结束执行时间："+historicProcessInstance.getEndTime());

System.*out*.println("执行时长："+historicProcessInstance.getDurationInMillis());

}

}

dao

根据 businesskey查询采购单信息。

使用逆向工程 生成的 mapper

service

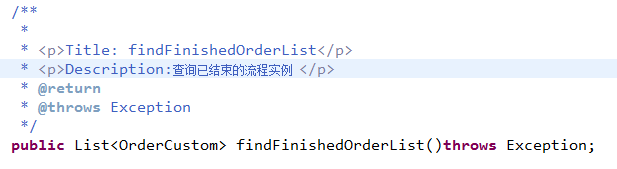
接口功能：查询已结束 的采购流程

接口参数：查询条件，如果此接口可以查询所有业务流程实例，查询条件必须将流程定义的key(可能包括：采购流程、请假流程。。)传入

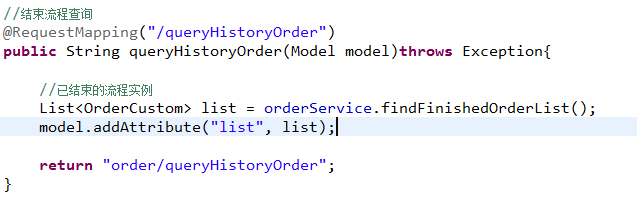
接口实现 ：

通过HistoryService查询已结束的流程实例，从act\_hi\_procinst表中查询结束 的流程实例

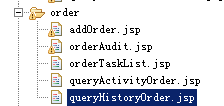
得到businessKey关联查询业务系统 数据



action



页面



## 采购单统计的问题

分析

进行采购单统计，按分类统计采购金额总数，统计的对象是结束的采购单。

如果只从业务系统 查询数据，如下，无法知道哪些采购单已结束：

Select sum(price) from pur\_bus\_order where order by XXX

如何处理？

* 处理方法1：

通过业务系统 表pur\_bus\_order和activiti的表act\_hi\_procinst进行关联查询。

通过如下的sql查询出结束的采购流程：

SELECT

\*

FROM

pur\_bus\_order,

act\_hi\_procinst

WHERE pur\_bus\_order.id = act\_hi\_procinst.BUSINESS\_KEY\_

AND act\_hi\_procinst.END\_TIME\_ IS NOT NULL

上边的sql要和activiti去关联，必须对activiti的数据库表非常清楚，如果数量大性能有问题。

* 处理方法2 :

最直接的方法，只从业务系统 中查询采购单信息，并进行统计。这种方法推荐的使用，因为统计的数据是业务数据，只能从业务系统 统计，为了activiti和业务系统 分工明确，业务系统 只负责业务数据。

如果使用方法2，只从pur\_bus\_order表查询采购单信息，此表中必须要有采购单结束标识。

设想，如果使用status字段设置是否结束，实现 上边 的方法2需求。

实现方法：

在activiti流程完成后，通过activiti帮助，在流程完成后，将status标识存储。

1》方法1 :通过TaskListener

在流程最后一个节点定义一个TaskListener，此监听器在任务完成时执行，在监听器中更新pur\_bus\_order表中status字段值 为complete（已完成）。

2》方法2:通过ExecutionListener

在endevent结点上添加ExecutionListener监听器。

监听事件选择end：



监听器在endevent结点完成后执行，执行的内容是更新pur\_bus\_order表中status字段值 为complete（已完成）。

在任务完成更新status实现

使用上边ExecutionListener开发。

定义一个ExecutionListener，实现ExecutionListener接口。

监听器实现步骤：

1. 从execution中获取businessKey
2. 根据businessKey即采购单id更新pur\_bus\_order表的status字段为complete。

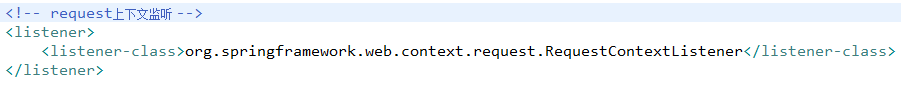
监听器中获取mapper问题

监听器的实例化不是通过spring容器，由activiti自己控制。

必须在监听 器获取spring容器。

通过servletContext获取spring容器。

第一步：在web.xml定义 request的上下文



第二步：通过request上下文得到servletContext，从而得到applicationContext

HttpServletRequest request = ((ServletRequestAttributes)RequestContextHolder.*getRequestAttributes*()).getRequest();

*applicationContext* = WebApplicationContextUtils.*getRequiredWebApplicationContext*(request.getServletContext());

将上边的代码封装到工具类中。

**public** **class** ApplicationContextUtils {

**private** **static** ApplicationContext *applicationContext*;

**public** **static** ApplicationContext getApplicationContext(){

**if**(*applicationContext* == **null**){

HttpServletRequest request = ((ServletRequestAttributes)RequestContextHolder.*getRequestAttributes*()).getRequest();

*applicationContext* = WebApplicationContextUtils.*getRequiredWebApplicationContext*(request.getServletContext());

}

**return** *applicationContext*;

}

}

#### ProcessCompleteListener实现

##### 编写监听器：

**public** **class** ProcessCompleteListener **implements** ExecutionListener {

// 通过工具类获取spring容器

**private** **static** ApplicationContext *applicationContext* = ApplicationContextUtils

.*getApplicationContext*();

@Override

**public** **void** notify(DelegateExecution execution) **throws** Exception {

// execution是流程实例 的代理对象

// 业务标识，即采购单id

String businessKey = execution.~~getBusinessKey~~();

// 通过pur\_bus\_order 的mapper更新pur\_bus\_order

//从spring容器中得到mapper

PurBusOrderMapper purBusOrderMapper = (PurBusOrderMapper) *applicationContext*.getBean("purBusOrderMapper");

//根据 采购单id更新status状态值为complete

PurBusOrder purBusOrder\_update = **new** PurBusOrder();

purBusOrder\_update.setId(businessKey);//更新记录的id

purBusOrder\_update.setStatus("complete");//更新status状态值为complete

purBusOrderMapper.updateByPrimaryKeySelective(purBusOrder\_update);

}

}

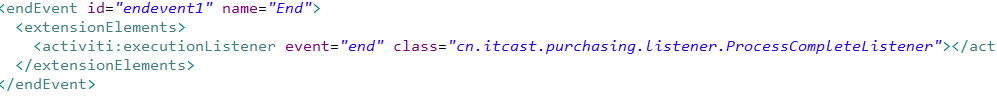
##### 在pbmn文件中配置监听器：

在流程图的endevent结点配置监听器：

监听器事件类型为end

查询bpmn文件：

监听器配置如下：



### 测试步骤

第一步：部署流程定义文件

第二步：启动一个流程实例

第三步：查询待办任务

第四步：办理任务

第五步：流程结束，观察pur\_bus\_order表的status字段值是否为complete，如果是表示测试通过

pur\_bus\_order表的status字段值



### 小结

根据业务系统需求，需要通过activiti将流程的运行数据写入业务系统 表。

通常使用监听器方法来完成。

明确需求：采购单统计，统计源数据特点：1>标识采购单是否完成 2>数量很大

最好将源数据放在一张表，从一张表进行统计，这样单表查询统计速度很快。

因为统计的源数据是业务数据，所以必须从业务系统 统计,需要通过activiti将流程的运行数据写入业务系统 表.

什么时候用关联查询？

如果需要关联查询出的数据量很大，不能在for循环中一个一个关联查询的！！！由于数据量大，需要关联其它表查询速度慢。

如果需要关联查询出的数据量小，可以for循环（一页的数据）中关联查询，并且通过主键或索引查询，性能没有影响。

# 查询历史任务

## 需求

查询某个流程下的历史任务。

查询内容：任务id、任务名称、任务负责人、任务标识

## 分析

使用HistoryService查询历史任务。

查询历史 任务时，如果指定assignee只查询某个用户所办理的历史任务。

Api代码如下：

**public** **void** queryHistoryTask(){

HistoryService historyService = processEngine.getHistoryService();

//创建查询对象，用于查询历史 任务

HistoricTaskInstanceQuery historicTaskInstanceQuery = historyService.createHistoricTaskInstanceQuery();

//设置查询条件

//设置taskAssignee只查询某个用户的历史 任务

//historicTaskInstanceQuery.taskAssignee(taskAssignee);

//指定 流程定义 key，只查询该流程下所有流程实例 所有历史 任务

String processDefinitionKey = "purchasingflow";

historicTaskInstanceQuery.processDefinitionKey(processDefinitionKey);

//指定 流程实例 id,只查询该 流程实例 执行的历史 任务，流程实例 的id可以完成也可以未完成的

String processInstanceId = "1401";

historicTaskInstanceQuery.processInstanceId(processInstanceId);

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

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

System.*out*.println("-------------------------------");

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

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

System.*out*.println("任务标识：" + historicTaskInstance.getTaskDefinitionKey());

System.*out*.println("任务负责人：" + historicTaskInstance.getAssignee());

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

System.*out*.println("任务开始时间：" + historicTaskInstance.getStartTime());

System.*out*.println("任务结束时间："+historicTaskInstance.getEndTime());

//如果要关联查询业务数据，由于数量很少，通过businesskey关联 查询业务系统 表

//获取流程实例 idhistoricTaskInstance.getProcessInstanceId()

//通过流程实例 id查询实例 对象 ，获取 businessKey

//...

}

}

dao

根据 businesskey查询采购单信息。

使用逆向工程 生成的 mapper

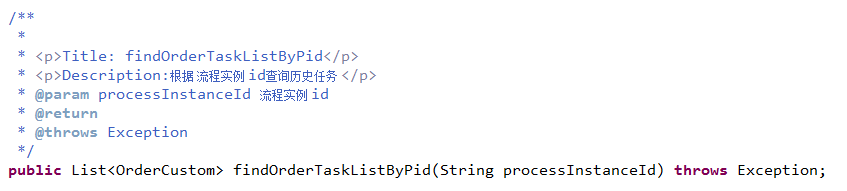
service

接口功能：根据流程实例 id查询历史任务

接口参数：流程实例 id

接口内容：

调用activiti的HistoryService查询历史 任务



**public** List<OrderCustom> findOrderTaskListByPid(String processInstanceId)

**throws** Exception {

// 创建查询对象，用于查询历史 任务

HistoricTaskInstanceQuery historicTaskInstanceQuery = historyService

.createHistoricTaskInstanceQuery();

// 设置查询条件

// 设置taskAssignee只查询某个用户的历史 任务

// historicTaskInstanceQuery.taskAssignee(taskAssignee);

// 指定 流程定义 key，只查询该流程下所有流程实例 所有历史 任务

String processDefinitionKey = "purchasingflow";

historicTaskInstanceQuery.processDefinitionKey(processDefinitionKey);

// 指定 流程实例 id,只查询该 流程实例 执行的历史 任务，流程实例 的id可以完成也可以未完成的

historicTaskInstanceQuery.processInstanceId(processInstanceId);

//添加排序，按照任务执行时间先后顺序

historicTaskInstanceQuery.orderByHistoricTaskInstanceStartTime().asc();

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

//即使这里只查询activiti中的数据，不关联查询业务的数据，也要单独定义 List<OrderCustom>，为了通过service和activiti和控制层隔离（解耦）

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

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

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

orderCustom.setTaskId(historicTaskInstance.getId());//任务id

orderCustom.setTaskName(historicTaskInstance.getName());//任务名称

orderCustom.setAssignee(historicTaskInstance.getAssignee());//任务负责人

orderCustom.setTaskDefinitionKey(historicTaskInstance.getTaskDefinitionKey());//任务标识

orderCustom.setTask\_startTime(historicTaskInstance.getStartTime());//任务开始时间

orderCustom.setTask\_endTime(historicTaskInstance.getEndTime());//任务结束时间

orderList.add(orderCustom);

}

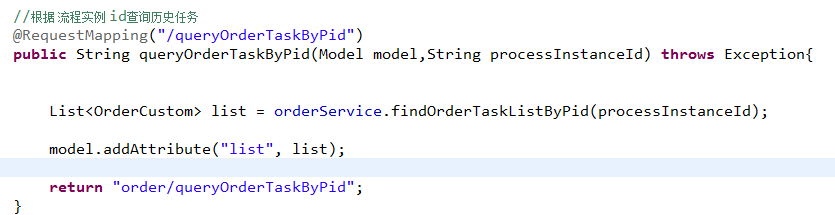
**return** orderList;

}

action

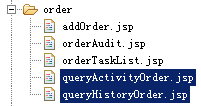
编写根据流程实例 id查询历史 任务方法：

接收参数：流程实例 id

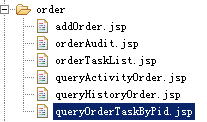


页面

修改当前运行流程页面和结束流程查询页面，添加“查看任务”连接，将流程实例 id传入上边的action方法。



编写历史任务列表页面：



## 查询某个用户所办理的历史任务

功能：我的历史任务

从session中获取当前用户id，设置到查询条件assignee中。

//设置taskAssignee只查询某个用户的历史 任务

//historicTaskInstanceQuery.taskAssignee(taskAssignee);

# 流程变量

## 什么是流程变量

在activiti在管理流程中，可能需要通过流程变量控制流程的执行。

**注意：流程变量只是用于控制流程的执行，而不是存储业务数据！！！**

## 流程变量作用域

Activiti的流程变量包括 global全局变量和local局部变量。

Activiti常用全局变量进行流程控制，因为local局部变量作用域小一般不用，可以查询历史的局部变量。

global全局变量：作用域是整体流程实例，如果流程实例结束，变量无效。

local局部变量: 作用域小，可以是一个任务(task)也可以是一个执行分支(execution)。任务或执行分支结束，local局部变量无效。

通过historyService查询历史流程变量值。

global全局变量和local局部变量，如下图：

流程实例001

orderPrice(采购金额)

流程实例002

orderPrice(采购金额)

创建采购单任务

orderPrice(采购金额)

创建采购单任务

orderPrice(采购金额)

审核采购单任务

orderPrice(采购金额)

审核采购单任务

orderPrice(采购金额)

不同的流程实例，global全局变量互不影响。

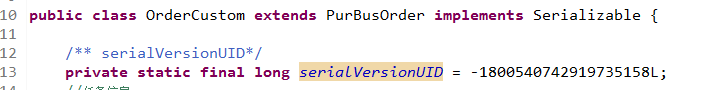
同一个流程实例，无法设置两个相同名称 global全局变量，后设置的相同名称 global全局变量会覆盖前边的变量值 。

任务中local变量，不同任务的local变量互不影响。Local变量名称可以和global全局变量相同。

## 流程变量的类型



注意：如果将pojo对象存储到流程变量中，必须实现序列化接口serializable，为了防止由于新增字段无法反序列化，需要生成serialVersionUID，如下：



注意：由于OrderCustom也继承了PurBusOrder类，PurBusOrder也必须实现序列化接口，否则 PurBusOrder中的属性无法反序列化。

总结：

1. 如果需要将pojo中的属性从流程变量获取（activiti进行反序列化），需要将属性所在pojo实现serializable接口。
2. 需要在pojo中设置serialVersionUID，如果不设置该版本id，日后在pojo中新增字段，如果之前将未新增字段的pojo对象存储在activiti的流程变量中，如果从流程变量中获取之前 的pojo对象，将报错。

## 流程变量的使用方法

第一步：设置流程变量

第二步：使用流程变量控制流程的执行

例子：

1. 在task结点的assignee通过UEL表达式使用流程变量

**${assignee}**：assignee就是一个流程变量名称

Activiti获取UEL表达式的值 ，即流程变量assignee的值 ，将assignee的值作为任务的负责人进行任务分配.

1. 在连线上的candition条件上通过UEL表达式使用流程变量

**${price>=10000}和${price<10000}**: price就是一个流程变量名称，uel表达式结果类型为**布尔类型**

如果UEL表达式是true，要决定 流程执行走向。

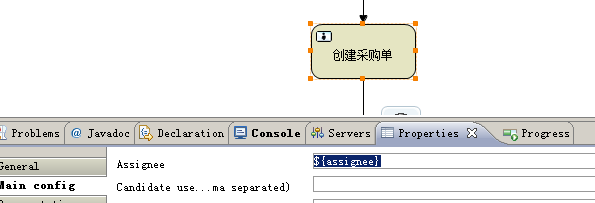
比如：如果采购金额大于10000元由总经理审核，否则由财务直接审核 。

## 设置全局变量

可以流程启动时设置流程变量（常用）

由于设置了全局变量，该流程启动后，在下边每个结点都可以使用该 变量。

在task结点的assignee通过UEL表达式使用流程变量



代码：

// 得到runtimeService

RuntimeService runtimeService = processEngine.getRuntimeService();

// 根据流程定义的key（标识 ）来启动一个实例，activiti找该key下版本最高的流程定义

// 一般情况下为了方便开发使用该方法启动一个流程实例。

String processDefinitionKey = "purchasingflow";

// 在流程启动时设置全局变量,第二个参数variables存储流程变量

Map<String, Object> variables = **new** HashMap<String, Object>();

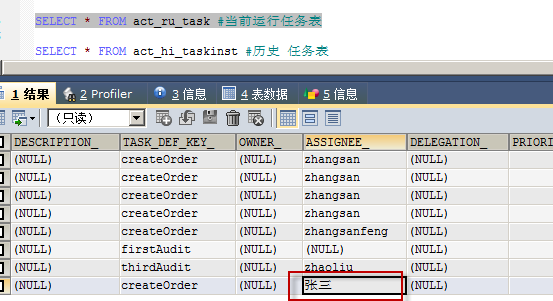
// 流程 变更名称是assignee，流程变量值是"张三"

variables.put("assignee", "张三");

ProcessInstance processInstance = runtimeService

.startProcessInstanceByKey(processDefinitionKey, variables);

测试：



在完成任务时设置流程变量（常用）

在完成任务时设置流程变量，在该 任务的后续结点可以使用该 变量，变量的作用域还是整个流程实例 。

//完成任务时设置流程变量

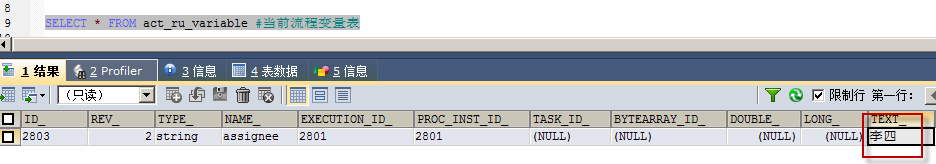
Map<String, Object> variables = **new** HashMap<String, Object>();

// 流程 变更名称是assignee，流程变量值是"张三"

variables.put("assignee", "李四");

taskService.complete(taskId, variables);

测试发现，在同一个流程实例下，同名的全局变量，后设置的会覆盖前边设置的：



通过当前流程实例 id设置

通过此方法可以在流程实例未结束时任意设置流程变量。

RuntimeService runtimeService = processEngine.getRuntimeService();

//设置单个 变量

//第一个参数：executionId是流程实例 的执行 id，通常使用流程实例 id，必须是当前正在运行的流程实例 id

//第二个参数：变量名

//第三个参数：变量值

runtimeService.setVariable("2001", "price", 10000);

//一次设置多个变量，第二个参数是map

//runtimeService.setVariables(executionId, variables)

注意：第一个参数executionId是流程实例 的执行 id，通常使用流程实例 id，必须是当前正在运行的流程实例 id

获取变量：

runtimeService.getVariable(executionId, variableName)

通过当前待办任务id设置

通过此方法可以在流程实例未结束时任意设置流程变量。

TaskService taskService = processEngine.getTaskService();

//设置单个 变量

//第一个参数：taskId是当前待办任务id，在act\_ru\_task存在

//第二个参数：变量名

//第三个参数：变量值

//activiti根据 任务id查询所属流程实例 id,存储act\_ru\_variable表。

taskService.setVariable("2902", "price", 10000);

//一次设置多个变量，第二个参数是map

//taskService.setVariables(taskId, variables)

activiti根据 任务id查询所属流程实例 id,将流程变量值存储act\_ru\_variable表，不管使用什么任务id，找到流程实例 id相同就会覆盖原来的变量。

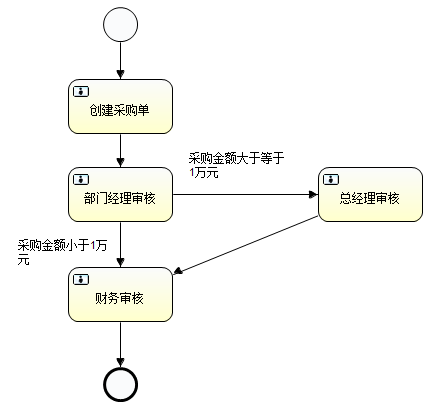
获取变量方法：

System.out.println(taskService.getVariable("3302", "price"));

## 全局变量测试

需求：

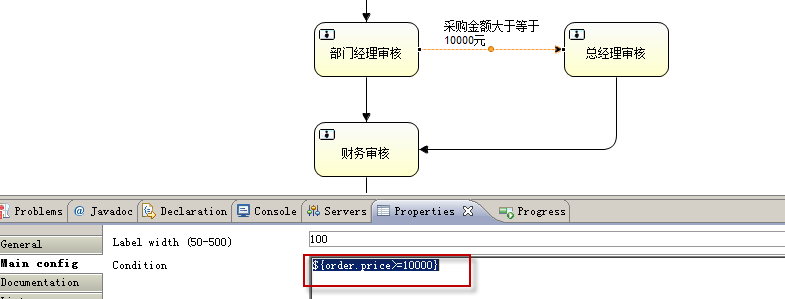
员工创建采购单，由部门经理审核，部门经理审核通过后一万元以下由财务直接审核，一万元以上先由总经理审核，总经理审核通过再由财务审核。



流程定义

在连线的candition上设置表达式，使用流程变量控制流程走向。

流程变量使用：OrderCustom pojo（包括 采购金额）



开发

什么时候设置流程变量？

采购流程中，在提交采购单时设置流程变量，因为提交采购单后采购单信息不再修改了。

使用完成提交采购单任务时设置流程变量。

// 完成任务时设置流程变量，使用pojo

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

orderCustom.setPrice(10000f);

Map<String, Object> varaibles = **new** HashMap<String, Object>();

varaibles.put("order", orderCustom);

taskService.complete(taskId, varaibles);

注意事项

注意：

1. 如果UEL表达式中流程变量名不存在，报错
2. 如果UEL表达式中流程变量值为空NULL，流程不按UEL表达式去执行，而流程结束 。（排他网关避免此问题）
3. 如果UEL表达式都不符合条件，流程结束 （排他网关避免此问题）
4. 如果连线不设置条件，会走flow序号小的那条线（排他网关避免此问题）

## 设置局部变量

在任务完成设置局部变量

在任务完成时设置局部变量，任务完成后，后续结点无法使用局部变量。

// 完成任务时设置流程变量，使用pojo

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

orderCustom.setPrice(10000f);

Map<String, Object> varaibles = **new** HashMap<String, Object>();

varaibles.put("order", orderCustom);

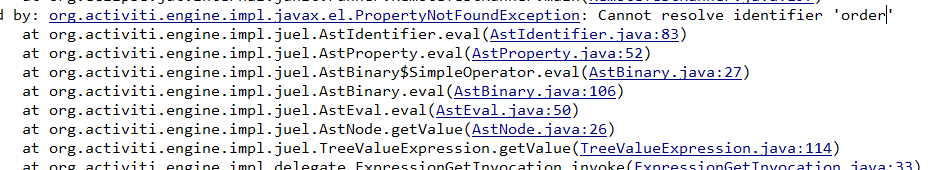
//设置局部变量

taskService.setVariablesLocal(taskId, varaibles);

//完成任务

taskService.complete(taskId);

当局部变量消失后，再使用该 变量会报错：



查询历史任务时查询流程变量

//查询历史 任务查询流程变量

historicTaskInstanceQuery.includeTaskLocalVariables();

//从activiti的流程变量取出pojo对象 ，经过反序列化

OrderCustom orderCustom = (OrderCustom) historicTaskInstance.getTaskLocalVariables().get("order");

System.*out*.println("历史 流程变量采购单信息："+orderCustom.getPrice());

**注意：使用流程变量不建议存储业务数据，因为通过activiti的api查询流程变量，特别是pojo变量，速度很慢。**

**流程变量要用于流程控制。**

通过当前任务id设置变量

//通过当前任务id设置局部变量

@Test

**public** **void** setVariableByTaskId(){

TaskService taskService = processEngine.getTaskService();

//设置单个 局部变量

taskService.setVariableLocal("5305", "price", 50000);

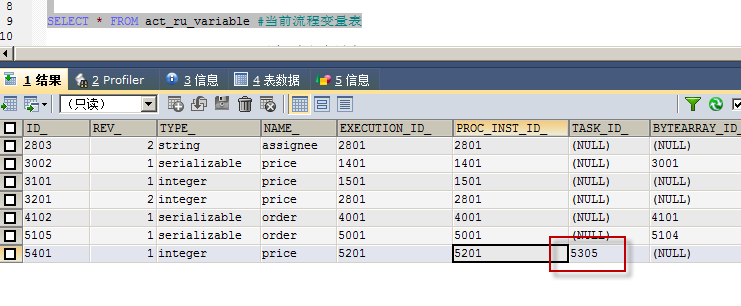
//一次设置多个局部变量

//taskService.setVariablesLocal(taskId, variables)

}

注意：任务id为当前待办理的任务id

测试发现：



## 流程变量数据表跟踪

SELECT \* FROM act\_ru\_variable #当前流程变量表

记录当前流程实例使用的流程变量

Type\_：变量类型

Name\_：变量名称

Execution\_id\_：流程实例执行id（global和local变量存储）

Proc\_inst\_id\_：流程实例 id（global和local变量存储）

Task\_id\_：流程变量所属的任务id（local变量存储）

BYTEARRAY\_ID\_:如果流程变量为pojo，字段存储引用act\_ge\_bytearray表的主键，在资源表act\_ge\_bytearray存储pojo流程变量的序列化信息。

Long\_和text\_:根据变量类型存储变量值

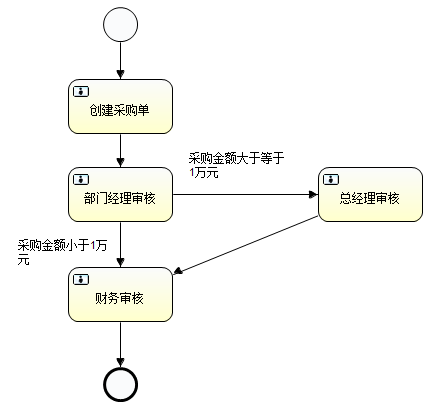
SELECT \* FROM act\_hi\_varinst #历史 流程变量表

记录流程执行所创建的所有流程变量

# 案例1

## 需求

员工创建采购单，由部门经理审核，部门经理审核通过后一万元以下由财务直接审核，一万元以上先由总经理审核，总经理审核通过再由财务审核。



## 流程定义

参考上边全局变量测试。

## 开发

采购流程中，在提交采购单时设置流程变量，因为提交采购单后采购单信息不再修改了。

使用完成提交采购单任务时设置流程变量。

修改提交 采购单service，在完成任务时设置流程变量。

修改orderService中saveOrderSubmitStauts提交采购单方法：

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

// 说明assignee是该任务的办理人，有权限完成

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

// 采购信息获取

// 根据 任务对象 获取流程实例 id

String processInstanceId = task.getProcessInstanceId();

// 查询流程实例 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查询采购单信息

PurBusOrder purBusOrder = purBusOrderMapper.selectByPrimaryKey(businessKey);

BeanUtils.*copyProperties*(purBusOrder, orderCustom);

//流程变量，值 为orderCustom即采购单信息

Map<String, Object> variables = **new** HashMap<String, Object>();

variables.put("order", orderCustom);

// 设置流程变量，值 为采购单信息

taskService.complete(taskId,variables);

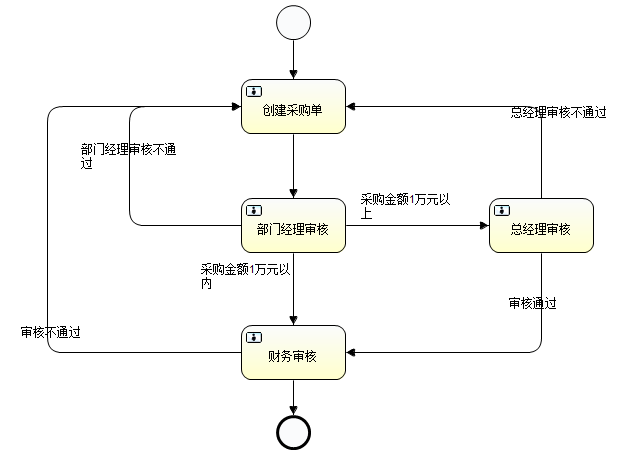
// System.out.println("完成任务：" + taskId);

}

# 案例2

## 需求

在上边的基础上添加审核不通过的连线。



需求如下：

在采购系统中实现流程审核不通过分支，功能如下：

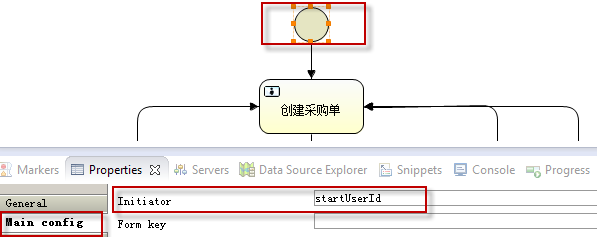
* 部门经理审核不通过由员工重新修改采购单
* 总经理审核不通过由员工重新修改采购单再提交
* 财务审核不通过由员工重新审核

## 流程定义

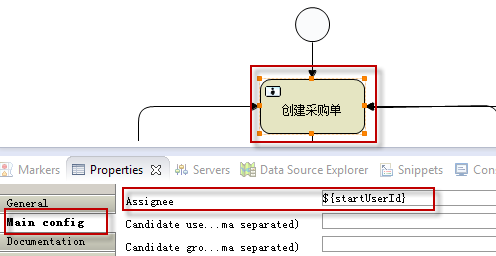
对审核不通的由流程发起人再提交采购单

设置流程发起人：

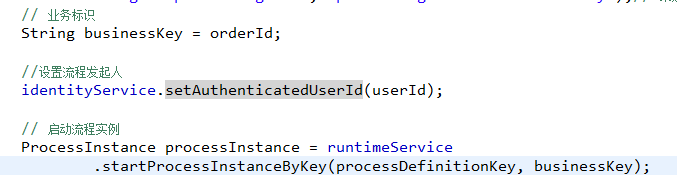
在开始结点，设置initiator（发起人）。



设置创建采购单的任务负责人为：



修改orderService中的saveOrder方法，在启动流程时设置任务发起人：



注意：

任务发起人设置在启动流程实例之前设置。

增加审核不通过的连线

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

注意：上边的firstAudit、secondAudit、thirdAudit分别存储三级审核信息。

## 开发

修改orderService中saveOrderSubmitStauts提交采购单方法：

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

// 说明assignee是该任务的办理人，有权限完成

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

// 采购信息获取

// 根据 任务对象 获取流程实例 id

String processInstanceId = task.getProcessInstanceId();

// 查询流程实例 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查询采购单信息

PurBusOrder purBusOrder = purBusOrderMapper.selectByPrimaryKey(businessKey);

BeanUtils.*copyProperties*(purBusOrder, orderCustom);

//流程变量，值 为orderCustom即采购单信息

Map<String, Object> variables = **new** HashMap<String, Object>();

variables.put("order", orderCustom);

// 设置流程变量，值 为采购单信息

taskService.complete(taskId,variables);

// System.out.println("完成任务：" + taskId);

}

修改 service中采购单审核方法：

设置流程变量：firstAudit、secondAudit、thirdAudit

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

// 说明assignee是该任务的办理人，有权限完成

Map<String,Object> variables = **new** HashMap<String,Object>();

//根据 auditType判断是几级审核

**if**(auditType.equals("firstAudit")){

//部门经理审核

variables.put("firstAudit", orderAuditCustom);

}**else** **if**(auditType.equals("secondAudit")){

//总经理审核

variables.put("secondAudit", orderAuditCustom);

}**else** **if**(auditType.equals("thirdAudit")){

//财务审核

variables.put("thirdAudit", orderAuditCustom);

}

//提交审核时，设置流程变量，变量值就是审核 信息

taskService.complete(taskId,variables);

// System.out.println("完成任务：" + taskId);

}