activiti第二天课堂笔记

# 复习

什么是工作流？

是通过计算机对业务流程进行自动化管理。实现多个参与者按照预定义的流程去自动化执行。

什么是activiti？

Activiti是一个工作流引擎（嵌入式执行），通常情况下activiti和业务系统集成在一块儿运行。是一个开源框架。

使用bpmn2.0标准去定义业务流程。

前身是jbpm4。

Activiti的使用步骤？

第一步：线下定义的业务流程。

遵循bpmn2.0标准（定义了业务流程的模型和符号）去定义业务流程，通常使用activiti官方提供eclipse-activiti-designer设计器。图形化的操作界面。

业务流程定义完成生成两个文件：.bpmn和.png

第二步：需要将流程定义文件(.bpmn和.png)部署到activiti中

方式1:单个文件部署（建议使用）

方式2：zip文件（.bpmn和.png压缩）部署

第三步：启动一个流程实例

流程定义：对业务流程按照bpmn2.0标准定义一个静态文件（xml格式）。

流程实例：参与者（可以是用户也可以是程序）按照流程定义内容发起一个流程，这就是一个流程实例。是动态的。

Activiti要对业务流程进行管理，要按照activiti的规则去启动一个流程实例。

流程实例启动了，流程开始按照.bpmn文件的内容去一步一步运行(动态)。

第四步：查询待办任务

当前用户查询当前自己的待办任务。

第五步：处理/办理任务

用户进行任务办理。

第六步：流程实例完成

当流程实例运行到最后一个结点，流程完成。

流程定义：

将流程定义文件(.bpmn和.png)部署到activiti中。

操作数据库表：

SELECT \* FROM act\_re\_deployment #流程部署表

SELECT \* FROM act\_re\_procdef #流程定义表（重要）

SELECT \* FROM act\_ge\_bytearray #资源表

act\_re\_deployment与act\_re\_procdef是一对多的关系。建议一次只部署一个流程，达到一对一关系，为了方便管理。

Activiti服务架构：

* Activiti.cfg.xml

Activiti的配置文件，spring格式的配置文件。

重点掌握：activiti和spring整合的配置文件。

* processEngineConfiguration

activiti的配置对象，获得processEngine

* processEngine

门面接口，通过此门面得到service

* service

|  |  |
| --- | --- |
| RepositoryService | activiti的资源管理类 |
| RuntimeService | activiti的流程运行管理类 |
| TaskService | activiti的任务管理类 |
| HistoryService | activiti的历史管理类 |
| IdentityService | activiti的用户身份管理类 |
| FormService | activiti的表单管理类 |
| ManagerService | activiti的引擎管理类 |

* activiti和spring整合的配置文件

processEngineConfiguration：**org.activiti.spring.****SpringProcessEngineConfiguration**

**要注入原系统的事务管理器，对activiti的事务和业务系统的事务统一管理。（activiti和业务系统集成开发要用）**

<bean id=*"processEngineConfiguration"* class=*"org.activiti.spring.SpringProcessEngineConfiguration"*>

<!-- 数据源 -->

<property name=*"dataSource"* ref=*"dataSource"* />

<!-- 使用spring事务管理器 Spring对业务系统的service和activiti的service使用一个事务管理器 -->

<property name=*"transactionManager"* ref=*"transactionManager"* />

<!-- 数据库策略 -->

<property name=*"databaseSchemaUpdate"* value=*"true"* />

<!-- activiti的定时任务关闭 -->

<property name=*"jobExecutorActivate"* value=*"false"* />

</bean>

processEngine：在spring配置文件中配置，将processEngineConfiguration注入。

<bean id=*"processEngine"* class=*"org.activiti.spring.ProcessEngineFactoryBean"*>

<property name=*"processEngineConfiguration"* ref=*"processEngineConfiguration"* />

</bean>

配置service：

<!-- 资源服务service -->

<bean id=*"repositoryService"* factory-bean=*"processEngine"*

factory-method=*"getRepositoryService"* />

# 教学计划

* 流程实例

如果启动一个流程实例。

数据库操作有哪些（掌握）

查询流程实例（掌握）

掌握：如何和业务系统集成起来去查询流程实例。

流程实例的挂起/激活（了解）

* 个人任务

指派任务（掌握）：

3种方式

查询待办任务（掌握）

掌握：如何和业务系统集成起来去查询任务。

办理任务（掌握）

掌握：如何和业务系统集成起来去办理任务。

* 案例：用activiti完成一个基本的采购流程（重点掌握）

如何和业务系统集成启动一个流程实例（重点）

如何和业务系统集成办理任务（重点）

# 流程实例

## 启动一个流程实例

参与者（用户）去发起一个流程，activiti规则需要先启动一个流程实例。

流程实例：参与者（可以是用户也可以是程序）按照流程定义内容发起一个流程，这就是一个流程实例。是动态的。

Api：

启动一个流程实例需要指定流程定义。

可以指定流程定义id

也指定流程定义的key(流程定义的key就是业务流程标识，key相同会自动找到最新的流程)

根据流程定义的key启动一个流程实例：

流程定义的key就是业务流程标识，在系统中配置文件（config/diagram/purchasingflow.properties）配置。

根据流程定义的key找最新版本的流程定义id。

启动流程实例之前要先部署流程定义。（如果流程定义已部署就不需要）

使用RuntimeService启动一个流程实例：

ProcessInstance processInstance = runtimeService

.startProcessInstanceByKey(processDefinitionKey);

流程实例启动后，流程实例就开始运行，首先流程运行到第一个结点。

代码如下：

**public** **void** startProcessInstance(){

//流程定义的key

String processDefinitionKey = "purchasingflow";

//runtimeService

RuntimeService runtimeService = processEngine.getRuntimeService();

//根据流程定义的key启动一个流程实例

ProcessInstance processInstance = runtimeService.startProcessInstanceByKey(processDefinitionKey);

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

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

System.*out*.println("当前流程实例的结点标识："+processInstance.getActivityId());

//activiti和业务系统集成后，需要将业务唯一标识存入activiti

//比如采购流程，就需要将采购单的id存储到流程实例表中。通过此businessKey关联查询业务系统信息

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

System.*out*.println("所属流程定义的id："+processInstance.getProcessDefinitionId());

}

数据库操作（重点）：

流程实例记录了**当前流程运行**的信息，应该从数据库中act\_ru\_\*找：

**SELECT \* FROM act\_ru\_execution #流程实例的执行表**

记录当前运行流程实例的执行信息。

如果流程实例运行完成了，此表的记录就会自动删除，**act\_hi\_procinst 历史表会保存记录**。

Id\_：流程实例执行id

PROC\_INST\_ID\_：流程实例的id

（理解）流程实例的id和流程实例执行id关系：

如果一个流程实例当前有多个分支正在执行，此时流程实例的执行表就有多条（流程实例的id和流程实例执行id不相同），始终有一条记录流程实例的id和流程实例执行id相等。

如果当前流程实例只有一个分支在执行，此表中有一条记录（流程实例的id和流程实例执行id相同）

BUSINESS\_KEY\_：业务标识，业务系统和activiti集成，需要将业务系统的表（采购流程是采购单表，请假流程是请假信息表）主键存入activiti，通过activiti去查询业务系统的数据（可以通过BUSINESS\_KEY\_查询业务系统数据）。

PROC\_DEF\_ID\_：流程定义id，流程实例所属流程定义

Act\_id\_：当前结点/活动id

**SELECT \* FROM act\_hi\_procinst #流程实例历史表**

记录了已经完成或未完成的所有流程实例

Id\_：流程实例执行id

PROC\_INST\_ID\_：流程实例的id

BUSINESS\_KEY\_：业务标识

PROC\_DEF\_ID\_：流程定义id

START\_TIME\_：开始时间

END\_TIME\_：结束时间，如果流程实例没有运行结束，此字段为null

DURATION\_：流程实例执行时长，如果流程实例没有运行结束，此字段为null

START\_USER\_ID\_：流程实例的发起人

## 查询流程实例

需求：查询系统中当前正在运行的流程有哪些？应该查询act\_ru\_execution表中数据，查询当前运行的流程实例信息。

流程定义和流程实例的图解：

流程定义：processDefinition

定义（静态）：.bpmn（xml格式）、png

部署到activiti数据库;

SELECT \* FROM act\_re\_deployment #流程部署表

SELECT \* FROM act\_re\_procdef #流程定义表

….

…..

流程实例：

processInstanece

参与者(用户)：

张三：创建一个采购单(101)表示发起一个采购流程（动态）

采购单101和这个流程实例对应上了。

完成了，这个实例就不存在了。

在历史表中有历史记录。

流程实例：

processInstanece

参与者(用户)：

李四：创建一个采购单(102)表示发起一个采购流程

采购单102和这个流程实例对应上了。

完成了，这个实例就不存在了。

在历史表中有历史记录。

流程实例：

processInstanece

参与者(用户)：

王五：创建一个采购单表示发起一个采购流程

api代码：

使用runtimeService。

查询需求：

流程实例id、所属流程定义id、当前运行活动标识、对应的采购单号、采购单名称、采购时间、采购金额

标红的在activiti的数据库中没有，需要通过businesskey查询采购单信息。

代码;

@Test

**public** **void** queryProcessInstance() {

// runtimeService

RuntimeService runtimeService = processEngine.getRuntimeService();

//创建一个查询对象

ProcessInstanceQuery processInstanceQuery = runtimeService.createProcessInstanceQuery();

//设置查询条件

//可以按照流程定义的key查询，只查询出某个业务流程的流程实例，不指定这个key查询出所有业务流程的流程实例

String processDefinitionKey="purchasingflow";

processInstanceQuery.processDefinitionKey(processDefinitionKey);

//取出查询记录

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

//分页

//processInstanceQuery.listPage(firstResult, maxResults)

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

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

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

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

System.*out*.println("当前流程实例的结点标识：" + processInstance.getActivityId());

// activiti和业务系统集成后，需要将业务唯一标识存入activiti

// 比如采购流程，就需要将采购单的id存储到流程实例表中。通过此businessKey关联查询业务系统信息

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

System.*out*.println("所属流程定义的id："

+ processInstance.getProcessDefinitionId());

//通过businessKey关联查询采购单表业务数据

//得到businesskey

String businesskey = processInstance.getBusinessKey();

//通过businesskey调用采购单mapper（业务系统的）mapper查询采购单信息

//.......

}

}

## Businesskey(业务标识)

启动流程实例时，指定的businesskey，就会在act\_ru\_execution #流程实例的执行表中存储businesskey。

Businesskey：业务标识，就是业务表的主键。业务标识和流程实例一一对应。业务标识来源于业务系统。

目的：就是根据业务标识来关联查询业务系统的数据。

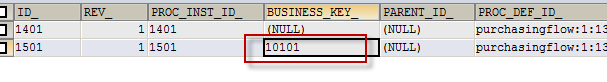
代码：

// 根据流程定义的key启动一个流程实例

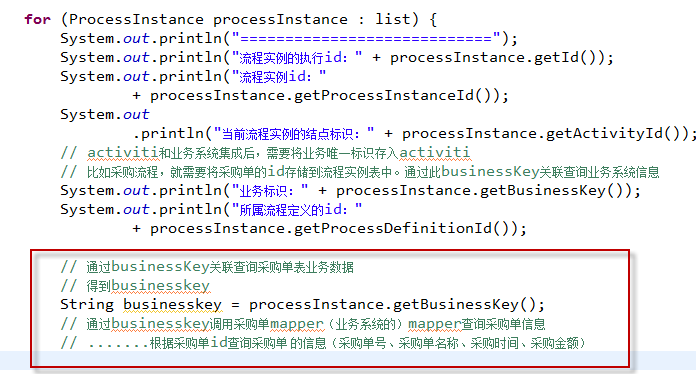
ProcessInstance processInstance = runtimeService

.startProcessInstanceByKey(processDefinitionKey, businessKey);

测试结果：

**

Businesskey的使用方法：



## 流程实例的挂起和激活

需求：如果修改流程，定义新的流程定义，将流程定义部署到activiti，旧流程还在运行。

对于旧流程操作方法：

可以删除旧的流程，使用级联删除，流程相当的数据全局删除（谨慎使用）

让已经运行的旧流程完成，不处理旧流程的运行。

暂停旧流程的执行，好处就是暂停了可以重新激活。

代码：

方式1:

针对流程定义挂起和激活流程，该流程定义下的所有流程实例全部挂起和激活。

如果流程实例挂起，流程实例运行暂停，如果用户继续完成任务执行流程实例，activiti会报错！！

需要指定流程定义id：processDefinitionId

挂起：

repositoryService.suspendProcessDefinitionById(processDefinitionId, **true**, **null**);

激活：

repositoryService.activateProcessDefinitionById(processDefinitionId, **true**, **null**);

通常情况下得到suspend是否暂停，根据这个状态执行暂停或激活操作

// 获得流程定义

ProcessDefinition processDefinition = repositoryService

.createProcessDefinitionQuery()

.processDefinitionId(processDefinitionId).singleResult();

//是否暂停

**boolean** suspend = processDefinition.isSuspended();

方式2：

针对某个流程实例执行挂起和激活。

需要流程实例的id：processInstanceId

挂起;

//如果激活则挂起

runtimeService.suspendProcessInstanceById(processInstanceId);

激活：

runtimeService.activateProcessInstanceById(processInstanceId);

通常情况下得到suspend是否暂停，根据这个状态执行暂停或激活操作

//根据流程实例id查询流程实例

ProcessInstance processInstance = runtimeService.createProcessInstanceQuery()

.processInstanceId(processInstanceId).singleResult();

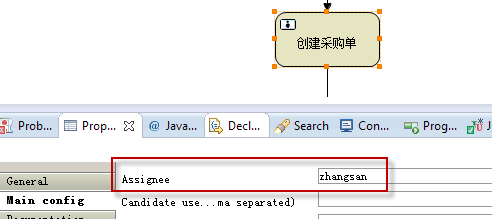
**boolean** suspend = processInstance.isSuspended();

# 个人任务

## 分配任务负责

固定分配方法

在流程定义时，在任务结点的properties视图中指定assignee（任务负责人）



任务分配：

在流程实例执行时，当运行流程到的某个task任务结点时，按照流程定义配置的assignee进行任务分配。

流程实例启动，执行到第一个结点。

数据库操作：

SELECT \* FROM act\_ru\_task #当前运行任务表（掌握）

记录当前运行的任务信息，当运行某个任务时向此表插入记录，如果该任务执行完成，该任务记录就从此表删除。

任务所属流程实例。(理解为流程实例下是任务)

Id\_：任务id

EXECUTION\_ID\_：流程实例执行id

PROC\_INST\_ID\_：流程实例id

PROC\_DEF\_ID\_：流程实例所属的流程定义(冗余存储，和流程实例id有联系)

Name\_：任务名称

TASK\_DEF\_KEY\_：任务标识

任务标识：在流程流程定义中Task的id。



ASSIGNEE\_：任务负责人，如果此任务当前有负责人，此字段就是负责人的id。

CREATE\_TIME\_：任务创建时间，流程实例执行到该任务结点时的时间。

SELECT \* FROM act\_hi\_taskinst #历史运行任务表：(掌握)

向当前任务表插入记录的同时，向历史任务表。

记录了当前运行的任务和已经结束的任务。

START\_TIME\_：开始时间

END\_TIME\_：结束时间

DURATION\_：任务执行时长

SELECT \* FROM act\_ru\_identitylink #当前流程实例参与者信息

记录流程实例运行时参与者信息（用户信息或组信息）

User\_id\_：用户id

Group\_id\_：组id（用户所属的组的id）

SELECT \* FROM act\_hi\_actinst #历史活动表

包括历史的任务、事件、网关等所有结点信息

小结：正式使用activiti通常不使用固定分配方法，一般情况下不会在流程定义阶段指定assignee任务负责人。

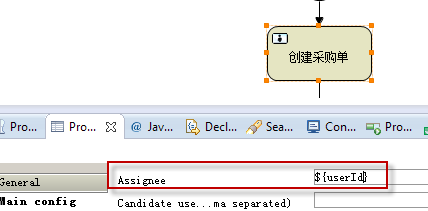
使用固定分配方法，系统可扩展性差。

uel表达式分配

Activiti使用UEL表达式，UEL是java EE6规范的一部分，UEL（Unified Expression Language）即统一表达式语言，activiti支持两个UEL表达式：UEL-value和UEL-method。

* UEL-value

定义uel表达式;



运行原理:

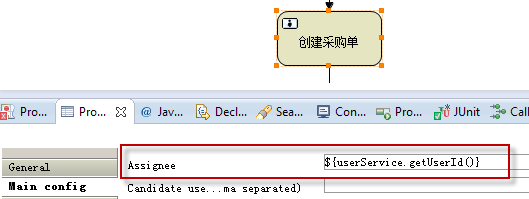
当流程实例运行到该任务结点，取出表达式${userId}值，userId是一个变量名，userId就是activiti流程变量中。

设置流程变量：

可以在启动一个流程实例时设置，也可以在完成任务时设置。

* UEL-method

定义：

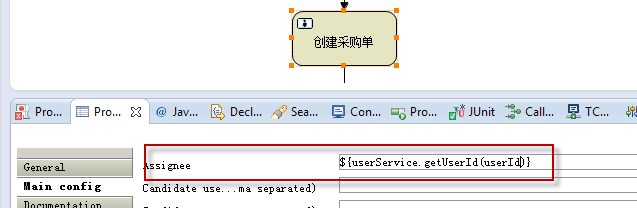


运行原理：

从spring的容器中找userService，调用getUserId方法，得到${userService.getUserId()}的值。

* UEL-method和Uel-value结合

定义：



运行原理：

从spring的容器中找userService，调用getUserId方法，时从activiti的流程变量找userId的值，向getUserid方法传入参数，得到整个表达式的值。

* 布尔结果类型

${price>=10000} ：流程变量price的值如果大于等于10000，${price>=10000}结果是true，否则 是false

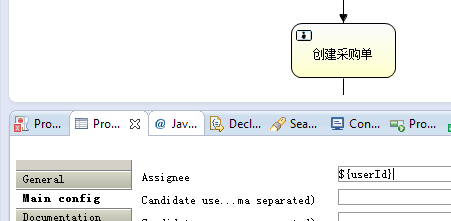
测试：

定义创建采购单结点的assignee为${userId}

在启动流程实例时设置该流程变量

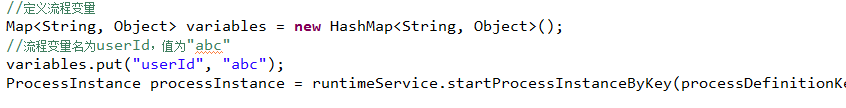
流程实例启动后，立即从流程变量中找userId的值，将此值当成创建采购单任务的负责人

流程定义：



部署流程定义：

启动流程实例：

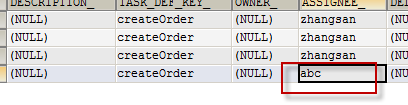


测试结果：

当前任务运行表：

SELECT \* FROM act\_ru\_task #当前运行任务表

Assignee：为abc

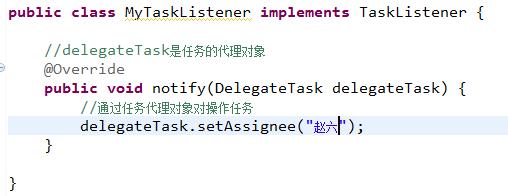


使用监听器分配(企业开发中常用)

定义：

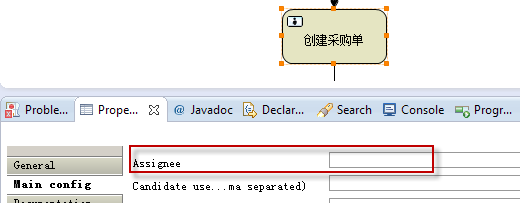
定义一个java类，实现任务监听器接口（TaskListener）。

设置任务负责：

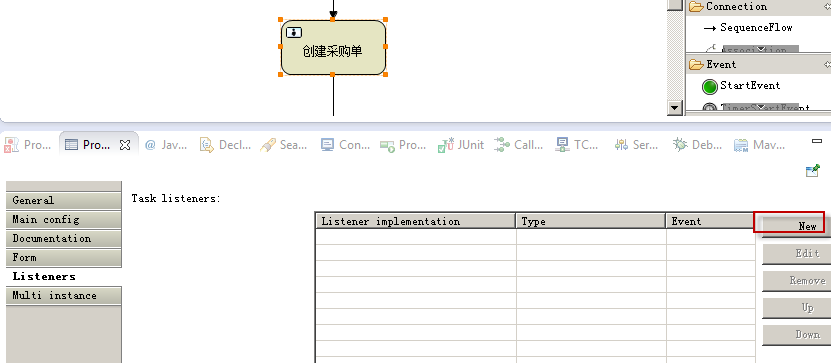


配置监听器：

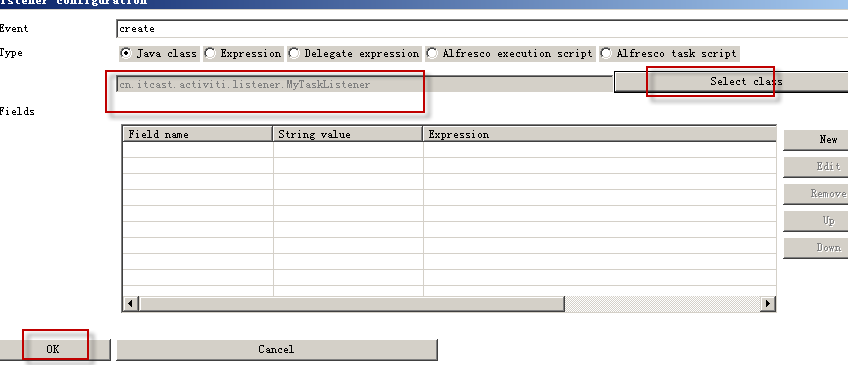
首先清除assignee的固定配置。



在任务结点的properties视图中配置listener：







设置监听类型：



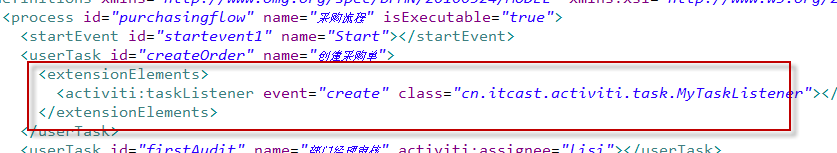
Create：任务创建时执行此监听器

Assignment：任务指派时执行此监听器

Complete：任务完成时执行此监听器

All：任务被activiti操作（create,assignment,complete）执行执行此监听器

通过查询bpmn文件，



文件中有配置的监听器。

测试;

查询当前任务表：

SELECT \* FROM act\_ru\_task #当前运行任务表

Assignee是通过监听器设置为“赵六”



小结：

在监听器nofity方法，可以扩展代码操作业务系统或activiti的数据库。此方法在企业开发中常用。！！！

## 任务查询

需求：当前用户登录系统后，查询当前该用户的待办任务。

采购流程需求：

需要查询：任务id、任务名称、任务标识、采购单号、采购单名称、采购金额

\*\*\*红色标识的字段在采购系统数据库中而不是在activiti数据库中。

如果查询采购单号、采购单名称、采购金额等业务数据需要根据businessKey查询。

方法如下：

从task对象中得到processInstanceId

根据processInstanceId得到processInstance对象

从processInstance对象中获得businessKey

Api代码：

使用TaskService查询当前待办的个人任务。

// 查询个人任务

@Test

**public** **void** queryPersonalTaskList() {

// 任务负责，正式开发系统当前用户从session中获得

String assignee = "zhangsan";

// 流程定义key

String processDefinitionKey = "purchasingflow";

TaskService taskService = processEngine.getTaskService();

RuntimeService runtimeService = processEngine.getRuntimeService();

// 创建任务查询对象

TaskQuery taskQuery = taskService.createTaskQuery();

// 设置查询条件,必须要指定任务的负责人

taskQuery.taskAssignee(assignee);

// 指定流程定义的key，只查询该业务流程下该用户的任务

taskQuery.processDefinitionKey(processDefinitionKey);

// 根据任务创建时间排序

taskQuery.orderByTaskCreateTime().asc();

// 根据任务创建时间排序(降序)

// taskQuery.orderByTaskCreateTime().desc();

// 查询该用户当前的待办任务

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

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

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

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

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

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

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

System.*out*.println("任务所属流程实例的id：" + task.getProcessInstanceId());

// 通过businessKey(业务标识)查询采购单号、采购单名称、采购金额

/\*\*

\* 从task对象中得到processInstanceId

\* 根据processInstanceId得到processInstance对象

\* 从processInstance对象中获得businessKey

\*/

// 流程实例的id

String processInstanceId = task.getProcessInstanceId();

// 根据流程实例的id查询流程实例对象

ProcessInstance processInstance = runtimeService

.createProcessInstanceQuery()

.processInstanceId(processInstanceId).singleResult();

// 业务标识(采购流程中就是采购单id)

String businessKey = processInstance.getBusinessKey();

// 根据businessKey即采购单id查询业务数据中采购单信息

// ........

System.*out*.println("businessKey:"+businessKey);

}

}

## 任务办理

需求：用户首先查看自己的待办任务，选定要办理的任务，进行任务办理。

采购流程需求：

当采购单提交后由部门经理审核，提交采购单就是一个任务办理的过程。

部门经理审核采购单，填写审核信息，将审核信息提交到业务系统的数据库，这个过程就是任务办理。

总经理审核采购单，填写审核信息，将审核信息提交到业务系统的数据库，这个过程就是任务办理。

财务经理审核采购单，填写审核信息，将审核信息提交到业务系统的数据库，这个过程就是任务办理。

特点：个人任务办理完成可以将业务流程向后推进一步。

Api代码：

使用TaskService完成任务。

首先需要指定任务id。

// 个人任务办理

@Test

**public** **void** completeTask() {

// 指定要办理任务id

String taskId = "2802";

// 任务负责人

String assignee = "zhaoliu";

TaskService takService = processEngine.getTaskService();

// 这里只指定任务id并没有指定任务负责人，只要调用complete，该任务就完成

// 所以说必须完成任务前校验

// 校验方法：

// 根据任务id和任务负责人assignee查询当前任务，如果查到说该用户有完成该任务权限，否则 没有权限

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

.taskAssignee(assignee).singleResult();

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

takService.complete(taskId);

System.*out*.println("完成任务！！！");

}

}

# 案例

## 需求

通过activiti完成采购流程：创建采购单---》部门经理审核采购单----》总经理审核采购单---》财务审核采购单

需求阶段：由需求分析人员进行分析，编写需求规格说明书，包括系统业务流程。

注意：需求阶段分析流程不涉及工作流。

设计阶段：根据需求文档，设计文档（详细设计文档包括类及方法的功能描述），在设计阶段需要由开发人员和需求分析人员一块儿讨论，讨论哪些业务流程由工作流来管理。

注意：针对复杂的，变化较大的业务流程建议使用工作流来管理。由于使用工作流要增加成本及工作量，对于简单的业务流程可以不用工作流。

设计出来系统功能：

创建采购单：

填写采购单信息

提前采购单，提交采购单表示该采购单不再修改了。

部门经理审核采购单:

进入审核页面

填写审核信息、提交

总经理审核：

进入审核页面

填写审核信息，提交

财务审核：

进入审核页面

填写审核信息，提交。

## 流程定义

设计人员确定哪些业务流程由工作流activiti来管理，需要使用activiti的流程设计器进行流程定义。（生成.bpmn和.png）。

如何进行流程定义？

1 需要考虑哪个功能和流程启动操作对应的.

2 需要考虑系统功能和流程定义中的结点对应关系问题。定义哪些系统功能可以将流程向后推进一步。

第一步：

根据设计阶段的系统功能，进行流程定义。

结点信息：

创建采购单

部门经理审核采购单

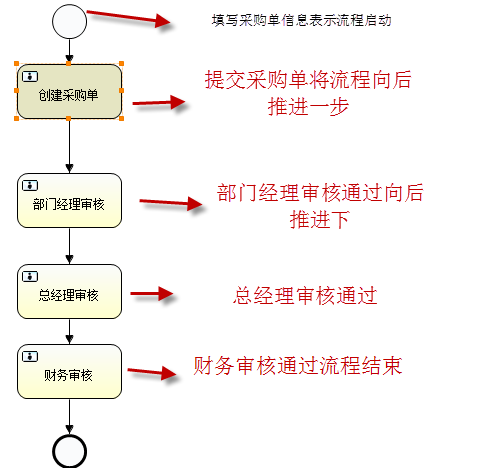
总经理审核采购单

财务审核采购单

第二步：

确定启动流程的功能

确定每个结点向后推荐流程对应的功能



## Activiti开发需要遵循的基本原则

**Activiti只负责流程管理，流程定义、流程执行自动向推进。**

**业务系统负责功能功能开发。比如：采购单创建、采购单审核。**

功能：

业务功能需要和activiti进行整合：

此功能执行是否表示启动一个流程？

如果此功能执行表示启动一个流程，应该在功能service方法中调用activiti的api启动一个流程。

此功能执行是否表示将流程向后推进一步？

如果此功能执行表示将流程向后推进一步，应该在功能service方法中调用activiti的api完成一个任务。

数据：

如果页面上需要流程相当的内容(比如：任务名称、任务办理人)，应该从activiti数据库取，调用activiti的service方法获取。

如果页面上需要业务系统的数据(比如:采购单名称，采购金额），应该从业务系统数据库取，调用业务系统的service获取。

## 创建采购单

需求：

企业中要进行采购（办公用品），按照企业采购流程，首先由员工创建采购单。

填写采购单信息功能：

创建采购单需要填写采购单名称、采购金额、采购内容等信息。

Dao：

功能：将采购单信息保存到数据表pur\_bus\_order中。

使用mybatis逆向工程生成pur\_bus\_order的mapper接口。



Service：

**Service接口定义的规则：**

**方法的业务参数尽量细化，且方法遵循单一职责。**

**描述：**

调用dao将将采购单信息保存到数据表pur\_bus\_order中

调用activiti的service启动一个流程实例。

**业务系统和工作流数据共享？共享businessKey和流程实例id。**

**将采购单id写入actviti中，**

**将流程实例id写入业务系统中采购单表。**

**达到目标：业务系统和activiti互相都能找到！！！！！**

**需要在采购单中添加一个字段processInstance\_id存储流程实例id!!**

**逆向工程生成的po类建议不要添加新的字段，除非特殊情况。**

为了提高系统扩展性，在原有po类扩充一个自定义的custom类。



保存采购单service代码;

@Override

**public** **void** saveOrder(String userId, OrderCustom orderCustom)

**throws** Exception {

// 采购单表的主键

// 使用uuid

String orderId = UUIDBuild.*getUUID*();

// 调用activiti的service启动一个流程实例

// 需要将采购单的id存储到activiti中

String businessKey = orderId;

//流程定义的key

String processDefinitionKey = ResourcesUtil.*getValue*(

"diagram.purchasingflow", "purchasingProcessDefinitionKey");

//启一个流程实例

ProcessInstance processInstance = runtimeService

.startProcessInstanceByKey(processDefinitionKey, businessKey);

//启动的流程实例的id

String processinstanceId = processInstance.getProcessInstanceId();

// 向业务系统数据库保存采购单信息

// 调用mapper保存采购单信息

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

// 将orderCustom对象的属性值拷贝到purBusOrder对象中，只要属性名相同就能拷贝

BeanUtils.*copyProperties*(orderCustom, purBusOrder);

//主键

purBusOrder.setId(orderId);

//流程实例的id

purBusOrder.setProcessinstanceId(processinstanceId);

//创建用户的id

purBusOrder.setUserId(userId);

purBusOrderMapper.insert(purBusOrder);

}

在applicationContext-service.xml配置service：



Action：

* 打开一个采购单创建页面

//打一个创建采购单页面

@RequestMapping("/addOrder")

**public** String addOrder(Model model)**throws** Exception{

**return** "order/addOrder";

}

* 填写信息提交采购单：

调用service保存采购单信息

// 保存采购单信息

@RequestMapping("addOrderSubmit")

**public** String addOrderSubmit(HttpSession session,OrderQueryVo orderQueryVo) **throws** Exception {

//从session获取用户

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

//保存采购单

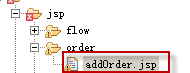
orderService.saveOrder(activeUser.getUserid(), orderQueryVo.getOrderCustom());

//转到采购单列表

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

}

页面



测试

测试步骤：

流程定义

部署流程定义

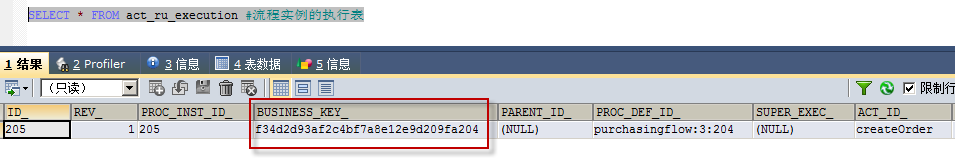
测试检查：

业务系统采购单表 pur\_bus\_order



将启的流程实例id保存到processInstance\_id字段。

Activiti流程实例表：SELECT \* FROM act\_ru\_execution #流程实例的执行表



将采购单id保存到business\_key\_字段。

查询待提交的采购单功能

需求：采购单创建完成后，不再修改了，提交采购单，提交后由部门经理开始审核。

查询待提交的采购单需求：

任务id、任务名称、任务标识、采购单id、采购单名称，采购金额。。

Dao：

采购单id、采购单名称、采购金额、采购时间：从业务系统的pur\_bus\_order表查询。

任务id：从activiti中查询。

Service：

查询待提交的采购单。

分析：

调用actviti的service查询当前用户待办任务。

从任务中获取流程实例id

根据流程实例id查询流程实例对象

再流程实例对象查询businessKey（采购单id）

根据采购单id从业务系统的pur\_bus\_order表查询采购单信息。

Service方法中创建一个单独的po类将activiti的数据和业务系统的数据整合到该类中。

**业务功能service**方法能否直接返回actviti的对象？

建议不要这么做，为了减少actviti和系统的耦合性。如果在service层将activiti的对象返回到action和页面，如果工作流引擎变了，就需要修改service、action、页面。

创建一个单独的po类，在service层将activiti和action隔离开了。

重点：哪些数据从activiti中查询，哪些数据从业务系统查询。

@Override

**public** List<OrderCustom> findOrderTaskList(String userId) **throws** Exception {

// 调用actviti的service查询当前用户待办任务。

// 从任务中获取流程实例id

// 根据流程实例id查询流程实例对象

// 再流程实例对象查询businessKey（采购单id）

// 根据采购单id从业务系统的pur\_bus\_order表查询采购单信息。

// 任务负责，正式开发系统当前用户从session中获得

String assignee =userId;

// 流程定义的key

String processDefinitionKey = ResourcesUtil.*getValue*(

"diagram.purchasingflow", "purchasingProcessDefinitionKey");

// 创建任务查询对象

TaskQuery taskQuery = taskService.createTaskQuery();

// 设置查询条件,必须要指定任务的负责人

taskQuery.taskAssignee(assignee);

// 指定流程定义的key，只查询该业务流程下该用户的任务

taskQuery.processDefinitionKey(processDefinitionKey);

// 根据任务创建时间排序

taskQuery.orderByTaskCreateTime().asc();

// 根据任务创建时间排序(降序)

// taskQuery.orderByTaskCreateTime().desc();

// 查询该用户当前的待办任务

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

//使用orderList将activiti和业务系统数据进行整合，使用orderList将activiti和action层隔离开了。

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

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

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

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

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

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

System.out.println("任务标识" + task.getTaskDefinitionKey());

System.out.println("任务所属流程实例的id：" + task.getProcessInstanceId());\*/

// 通过businessKey(业务标识)查询采购单号、采购单名称、采购金额

/\*\*

\* 从task对象中得到processInstanceId

\* 根据processInstanceId得到processInstance对象

\* 从processInstance对象中获得businessKey

\*/

// 流程实例的id

String processInstanceId = task.getProcessInstanceId();

// 根据流程实例的id查询流程实例对象

ProcessInstance processInstance = runtimeService

.createProcessInstanceQuery()

.processInstanceId(processInstanceId).singleResult();

// 业务标识(采购流程中就是采购单id)

String businessKey = processInstance.getBusinessKey();

// 根据businessKey即采购单id查询业务数据中采购单信息

//查询业务系统数据库

//根据主键查询表，for记录数是一页的数据。速度不慢。

PurBusOrder purBusOrder = purBusOrderMapper.selectByPrimaryKey(businessKey);

//orderCustom要将采购单和activiti的task进行整合

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

BeanUtils.*copyProperties*(purBusOrder, orderCustom);//将采购单所有信息全部拷贝到orderCustom

//任务id、任务名称、任务标识

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

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

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

orderCustom.setProcessinstanceId(task.getProcessInstanceId());//流程实例的id

orderCustom.setAssignee(task.getAssignee());

orderList.add(orderCustom);

}

//应该将activiti的数据和业务系统的采购单信息都返回

**return** orderList;

}

Action：

// 采购单列表

@RequestMapping("/orderTaskList")

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

**throws** Exception {

// 从session获取用户

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

List<OrderCustom> list = orderService.findOrderTaskList(activeUser.getUserid());

//将处理采购单数据list传到页面

model.addAttribute("list",list);

**return** "order/orderTaskList";

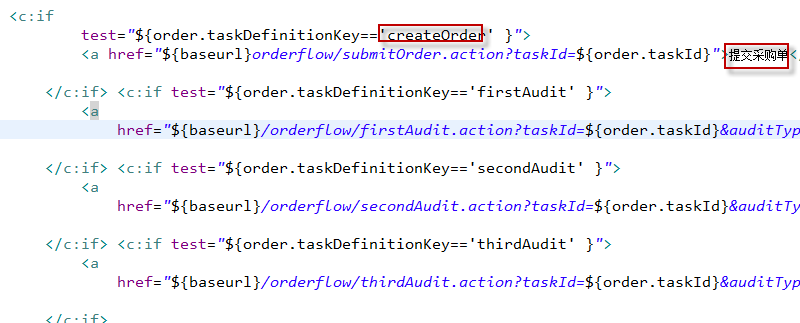
}

页面：



采购单处理的链接：

根据任务的标识显示是“提交采购单”、部门经理审核、总经理审核、财务审核。



测试：



提交某个采购单

需求：员工不再修改采购单，执行提交操作，采购单提交后由经理审核，提交后业务流程向后推进一步。

Dao：

提交采购单不需要更新业务系统中pur\_bus\_order状态字段，因为流程执行由Actviti进行控制。

Dao不需要开发。

Service:

提交采购单service：

分析：

调用activiti的service执行任务完成。

需要任务id才可以完成任务。

需要当前完成任务的用户id。

@Override

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

// 任务负责人

String assignee = userId;

// 这里只指定任务id并没有指定任务负责人，只要调用complete，该任务就完成

// 所以说必须完成任务前校验

// 校验方法：

// 根据任务id和任务负责人assignee查询当前任务，如果查到说该用户有完成该任务权限，否则 没有权限

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

.taskAssignee(assignee).singleResult();

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

taskService.complete(taskId);

System.*out*.println("完成任务！！！");

}

}

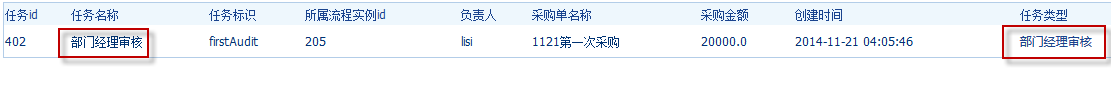
Action：

页面：

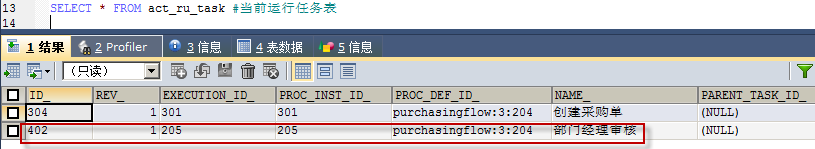
不需要

测试

使用lisi登录，查询采购单处理列表，



查询数据库



## 审核采购单

建议：

每个功能都需要写单独的service、action及页面，公用仅仅是代码级别。为了保证系统可维护可扩展性。

为了简化开发，教学方便将部门经理审核、总经理审核、财务审核做成一个方法。

打一个审核页面

在点击审核链接时，将任务id、采购单id、审核用户类型、传到action。

Action：

@RequestMapping("orderAudit")

**public** String orderAudit(Model model,String taskId,String orderId,String auditType)**throws** Exception{

//将任务id传到审核页面

model.addAttribute("taskId", taskId);

//采购单id

model.addAttribute("orderId", orderId);

//审核类型

model.addAttribute("auditType", auditType);

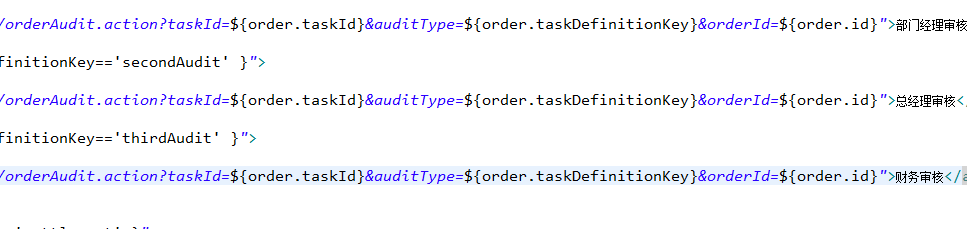
**return** "order/orderAudit";

}

页面：

修改采购处理列表页面：

将任务id、采购单id、审核类型传入审核action。



提交审核信息

Dao：

业务系统如何操作？

向业务系统采购单审核信息表pur\_bus\_order\_audit插入记录。

Activiti如何操作？

审核通过要完成任务，这时业务流程向后推进一步。

Service;

提交审核结果service：

向业务系统采购单审核信息表pur\_bus\_order\_audit插入记录（采购单审核信息）

审核通过调用activiti的service完成任务，需要任务id和用户id。

@Override

**public** **void** saveOrderAudit(String userId, String taskId, String orderId,

String auditType, OrderAuditCustom orderAuditCustom)

**throws** Exception {

// 调用mapper保存业务系统审核信息

orderAuditCustom.setId(UUIDBuild.*getUUID*());

orderAuditCustom.setUserId(userId);

orderAuditCustom.setOrderId(orderId);

orderAuditCustom.setAuditType(auditType);

purBusOrderAuditMapper.insert(orderAuditCustom);

//调用activiti完成任务

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

.taskAssignee(userId).singleResult();

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

taskService.complete(taskId);

System.*out*.println("完成任务！！！");

}

}

Action：

// 提交审核

@RequestMapping("/submitOrderAudit")

**public** String submitOrderAudit(HttpSession session,String taskId,String orderId,String auditType,OrderQueryVo orderQueryVo) **throws** Exception {

// 从session获取用户

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

//提交审核

orderService.saveOrderAudit(activeUser.getUserid(), taskId, orderId, auditType,

orderQueryVo.getOrderAuditCustom());

// 转到采购单列表

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

}

测试

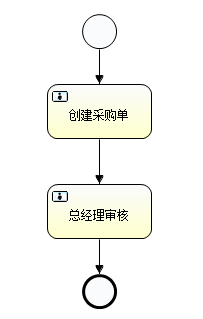
检查：

业务系统检查采购单审核表是否记录？

Activiti检查：审核通过，业务流程向后执行一步。

测试如果流程变更了，只需要修改activiti的流程定义，重新部署流程定义，业务功能代码不用改了。

因为只是流程变了，业务功能没有变。



重新进行部署流程定义。

重新走采购流程，由员工创建一个采购单。