# activiti学习总结

## 集成springboot

### 1.1 pom依赖

<**dependency**>  
 <**groupId**>org.activiti</**groupId**>  
<**artifactId**>activiti-spring-boot-starter</**artifactId**>

<**version**>7.0.0.Beta2</**version**>  
</**dependency**>

以下部分为xml转换bpmn模型需要的依赖  
<**dependency**>  
 <**groupId**>org.activiti</**groupId**>  
 <**artifactId**>activiti-modeler</**artifactId**>  
 <**version**>5.22.0</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>org.activiti</**groupId**>  
 <**artifactId**>activiti-bpmn-model</**artifactId**>  
 <**version**>7.1.0.M5</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>org.activiti</**groupId**>  
 <**artifactId**>activiti-bpmn-converter</**artifactId**>  
 <**version**>7.1.0.M5</**version**>  
</**dependency**>

### 1.2 springboot配置

此配置用于生成历史表格，让activiti具有完整的25张表

**activiti:** *#此配置用于生成history表* **history-level:** *full* **db-history-used: true**

注意事项：MAVEN打包配置需要指定资源配置文件的路径，在这个配置中最好将必须的资源配置路径写全。不然可能启动找不到路径，会被配置所忽略。

### 1.3SpringSecurity配置类

在配置类中配置相关springSecurity的配置

## 2.基本使用

### 2.1工具服务api

在集成springboot后，启动springboot会自动创建activiti表格，在spring容器中也会有相关服务类的实例。可以在代码中直接使用@Autowired直接注入使用。

@Autowired  
**private** RepositoryService **repositoryService**;

基本的服务API有以下几大类：

RepositoryService 管理流程定义

RuntimeService 执行管理，包括启动、推进、删除流程实例等操作

TaskService 任务管理

HistoryService 历史管理(执行完的数据的管理)

### 2.2基本的流程增删改查等

都用到repositoryService

#### 2.2.1新增

有多种部署方式，这里是文件方式，常用的主要以xml字符串进行部署。

特别注意：在使用文件以外的方式部署时，例如xml字符串，要对xml字符串的格式进行调整严格符合activiti保存的格式，对于流程定义ID的属性格式必须为Process\_\*，赋予文件名字时格式为\*bpmn20.xml

*/\*\*  
 \* 自测:流程定义部署  
 \* repositoryService  
 \* 关联的表：act\_re\_deployment流程部署信息表,act\_re\_procdef流程定义信息表，acr\_ge\_bytearray文件字节流保存表  
 \*/*Deployment deployment = **repositoryService**.createDeployment().addClasspathResource(**"test/diagram1.bpmn"**)  
 .name(**"文件部署流程"**)  
 .deploy();

#### 2.2.2编辑

activiti自身没有对流程编辑的api，我们直接使用在流程定义有具体的一个实例前，进行先删除，再创建新的流程来覆盖，processkey保持不变。

如果对于已经有流程实例开启需要对流程定义做编辑处理的，目前根据业务进行具体调整，涉及到activiti删除会报错，也可以进行强制删除，不过会对所有已经启动的流程实例进行删除，所有运行节点不复存在。

#### 2.2.3删除

*/\*\*  
 \* 删除流程定义 act中的物理删除  
 \* 定义信息，部署信息和字节流数据信息三张表信息被删除  
 \* 注意：一个流程没有安全审批结束时，不能删除流程的部署以及定义信息，会失败。  
 \* 如果要强制删除：repositoryService.deleteDeployment("48618084-3216-11ea-a230-a86daae9e0ea",true);  
 \* 之前未完成的和已完成的流程任务活动节点全部被删除，再删除基本的部署和定义信息等。  
 \*/***repositoryService**.deleteDeployment(**"18f9e3da-4588-11ea-87bc-a86daae9e0ea"**);*//流程部署的ID*

#### 2.2.4查询

activiti提供多种条件的高级查询，并可自带分页效果

特别注意：activti自身查出来的对象集合，不能直接返回给前端，需要用业务实体类进行转换，所以一般直接查业务实体类的表，前提是业务实体类的表与activiti高度同步，具体方式查询以实体表的关联设计为准

#### 2.2.5业务关联的注意事项

查询相关的业务关联：

一种方式是在业务表中存放activiti的唯一判断标准字段来反向查询activiti表，第二种方式直接在activiti表中的businessKey字段中加入业务表主键。

### 2.3流程的挂起

流程的挂起取决于属性suspended。

#### 2.3.1全部挂起

修改流程定义的suspended则可以控制该定义下所有流程实例的暂停，如果在暂停期间操作流程的继续运行，activiti会报错不能进行暂停的流程实例。

激活：

repositoryService.activateProcessDefinitionById(processDefinitionId,

**true**, **null**);

挂起：

repositoryService.suspendProcessDefinitionById(processDefinitionId,

**true**, **null**);

NULL值为Date数据，一般不需要加入时间，直接为NULL

#### 2.3.2流程实例的单独挂起

单独激活：

runtimeService.activateProcessInstanceById(processInstanceId);

单独挂起：

runtimeService.suspendProcessInstanceById(processInstanceId);

注意所用到的api，一般具体到实例的都是runtimeService

### 2.4个人任务

个人任务代表此任务节点归属于一个人办理。

#### 2.4.1固定分配

固定分配直接在bpmn文件属性框中直接指定个人的唯一标识，第二种以xml部署时直接修改userTask标签中的activiti:assignee的属性值。

#### 2.4.2表达式分配

基本格式${\*}，根据流程变量来进行赋值，流程变量在JAVA代码中用API进行配置。

##### 2.4.2.1流程变量

分为全局流程变量和局部流程变量：

1.全局流程变量为所有节点都可以获取使用，例如在网关节点进行条件值判断时，对上个节点设置的全局流程变量进行取值。

局部流程变量为单个节点的流程变量，其他节点不能获取。

2.流程变量的API设置，可以从启动具体的实例时，完成任务时的API多加一个变量参数来进行设置，也可以直接拿到实例或者其中一个任务进行设置。全局与局部的区别在于API后方是否有local。

3.流程变量的数据类型包含我们常用的多种类型，一般设置为字符串类型，也可以设置为对象类型，在表达式中采用”对象.方法/属性”的方式取值。

4.流程变量以及表达式同样可以用在分支线判断条件上，进行>,<,=,&&,||等逻辑运算符的运算。

注意事项：在流程变量的取用中，做到与JAVA相同的先声明再取用，如果流程变量的类型为对象类型，那么该对象对应的类应该实现序列化接口。业务中可在xml字符串中直接设置书写变量取值表达式也可以通过JAVA方式设置，简单的用xml，复杂的用JAVA。

### 2.5组任务

相当于候选人池，在不确定由哪位办理人办理时，设置多个候选人从中选择一个办理人，办理人需要拾取任务，才可在自己的任务中进行办理该任务，也可以转交和回退任务，将任务回退到可供所有候选人选择的状态，或者指定转交给候选人池中的其中一位办理人。

#### 2.5.1分配

activiti:candiateUsers代表候选人，可用逗号间隔，可用${}表达式用流程变量赋值集合。

#### 2.5.2拾取任务

taskService.claim(taskId, userId);

不是候选人也可以拾取任务，结合业务时最好进行判断区分，当任务被拾取后，其他候选人不能查询到该任务。

#### 2.5.3退回任务

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

将办理人设置为Null即可归还任务

#### 2.5.4交接/转交任务

taskService.setAssignee(taskId, candidateuser);

将办理人设置为另一办理人即可转交任务，转交时先归还任务。

### 2.6网关

#### 2.6.1排他网关



只能选择其中一个分支运行，如果两个分支都为true，则选择ID号较小的运行。如果都不满足条件没有分支可走，则会报错结束。设置条件时要注意

#### 2.6.2并行网关



直接运行所有分支，在分支运行都结束后需要再次汇聚在一个并行网关下，并行网关不会进行条件判断，哪怕分支上有条件。

#### 2.6.3包含网关



综合排他网关和并行网关，只要有一个分支满足条件判断，就会运行，同样需要聚合。

### 2.7任务的并行与串行

在activiti:collection中配置人员集合，一般是通过流程变量配置。

elementVariable为该集合的遍历变量，例如assigneeID

assignee配置${assigneeID},对变量进行取值

#### 2.7.1并行

isSequential="false"代表并行

通过以上配置，并行任务开始时会创建对应子任务，集合人员会同时接收到任务，并在所有任务完成后才能跳入下一节点，可以设置完成条件：

completionCondition：

${nrOfCompletedInstances/nrOfInstances >= 0.6 }

代表完成百分之六十则通过。

#### 2.7.2串行

任务依次按照集合人员顺序完成。

## 3.activiti7新特性

1.只要注入 ProcessRuntime 的实现对象，就可以实

现流程定义信息的操作，必须将SpringSecurity加入进来。

1. TaskRuntime是对于 TaskService的一个封装，操作任务相关业务。

## 关键业务的实现

*/\*\*  
 \* 相关业务需求  
 \* 1.任务驳回，直接结束流程  
 \* 2.自由流程：审批者指定下一步任务的审批者  
 \* 3.自由会签流程：申请人申请时指定会签候选人  
 \*/  
  
/\*\*  
 \* 1.任务驳回：结束单独的流程实例  
 \* (还有一种方法是通过每条执行路径的一个全局流程变量进行判断，  
 \* 每个审批任务去修改这个变量，系统找不到通行路径会自动结束流程实例的运行，  
 \* 较为麻烦)  
 \*/*@Test  
**public void** test8(){  
 *//拿到该流程实例* **securityUtil**.logInAs(**"salaboy"**);  
 ProcessInstance processInstance = **runtimeService**.createProcessInstanceQuery()  
 .processInstanceId(**"a4f18b1f-48a8-11ea-ac70-a86daae9e0ea"**)  
 .singleResult();  
 *//删除流程实例，添加删除理由，并在history中更新delete\_reason* org.activiti.api.process.model.ProcessInstance instance = **processRuntime** .delete(**new** DeleteProcessPayload(processInstance.getProcessInstanceId()  
 , **"驳回任务"**));  
 *//删除流程实例后只会在历史表中存留实例信息，runtime表中以及任务表中信息相关等删除* System.***out***.println(instance.getId());  
 System.***out***.println(instance.getStatus());  
  
}  
  
*/\*\*  
 \* 2.自由流程：审批者指定下一步任务的审批者  
 \* 流程定义创建时审批者assignee为动态设置${\*}，此后在每一个审批者审批时进行流程变量添加，  
 \* 第一个审批者需要默认固定指定一个，或者在流程启动时设置一个，  
 \* 则可指定下一个审批者(半自由流程在前端选择查询权限树限制选择即可)  
 \*/*@Test  
**public void** test9(){  
 **securityUtil**.logInAs(**"salaboy"**);  
 org.activiti.engine.task.Task task = **taskService**.createTaskQuery().taskAssignee(**"zhAngSan"**).singleResult();  
 *//完成任务的同时设置下一个节点的流程变量指定下一个节点的审批人* HashMap<String, Object> map = **new** HashMap<>();  
 map.put(**"user2"**,**"liSi"**);  
 **taskService**.complete(task.getId(),map);  
}  
  
*/\*\*  
 \*3.自由会签流程：  
 \* 在前端设置任务并行，并设置assignee动态取值为${demo1},  
 \* 会签人集合collection设置流程变量${\*},  
 \* 集合遍历变量elementVariable为demo1供assignee取值；  
 \* 这里会签采用并行，创建实例后，所有集合的人员会同时收到任务，并在全部完成后进入下一个任务节点，  
 \* 串行流程则为集合人员依次办理任务。  
 \* (抢签流程用候选人拾取任务完成，一个人拾取任务后并办理后直接跳入下一个节点)  
 \*/*@Test  
**public void** test10(){  
 org.activiti.engine.task.Task hq1 = **taskService**.createTaskQuery().taskAssignee(**"hq1"**).singleResult();  
 org.activiti.engine.task.Task hq2 = **taskService**.createTaskQuery().taskAssignee(**"hq2"**).singleResult();  
 org.activiti.engine.task.Task hq3 = **taskService**.createTaskQuery().taskAssignee(**"hq3"**).singleResult();  
 *//完成各自的任务,可设置完成任务比例，如不设置则只能等待所有子任务完成后，才会跳入下一节点* **taskService**.complete(hq1.getId());  
 **taskService**.complete(hq2.getId());  
 **taskService**.complete(hq3.getId());  
}