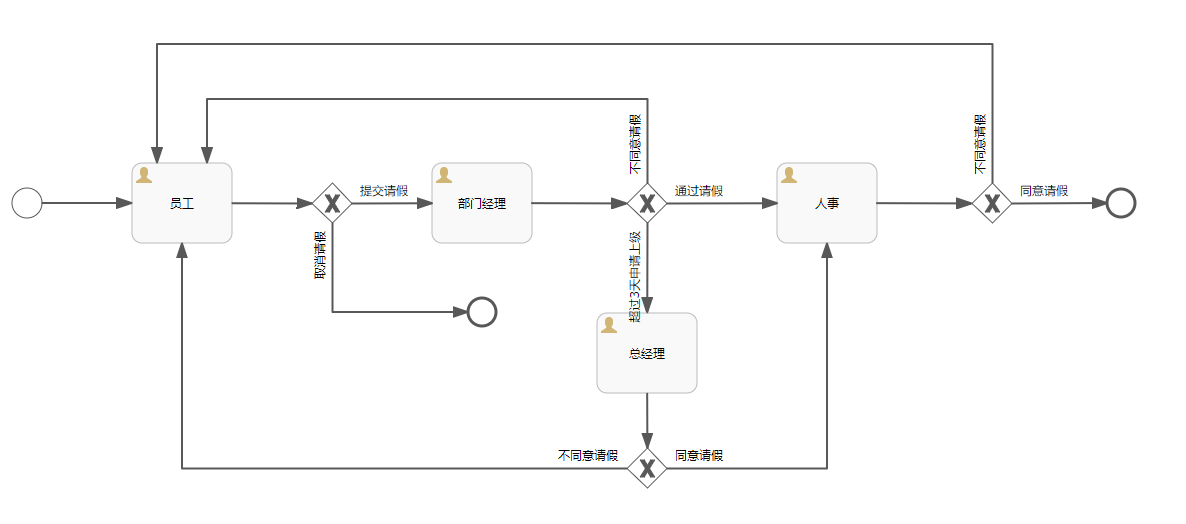
目录

1. Activiti简介
2. Springboot2.X + Activiti5.22集成在线编辑器
3. 核心API
4. 请假流程demo



1. Activiti简介：

工作流(Workflow)：

就是“业务过程的部分或整体在计算机应用环境下的自动化”，它主要解决的是“使在多个参与者之间按照某种预定义的规则传递文档、信息或任务的过程自动进行，从而实现某个预期的业务目标，或者促使此目标的实现”。

工作流管理系统(Workflow Management System, WfMS)：

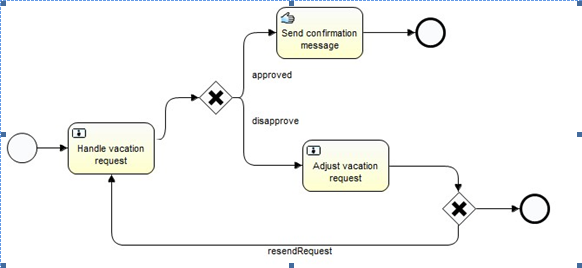
是一个软件系统，它完成工作量的定义和管理，并按照在系统中预先定义好的工作流规则进行工作流实例的执行。工作流管理系统不是企业的业务系统，而是为企业的业务系统的运行提供了一个软件的支撑环境。

工作流管理联盟(WfMC，Workflow Management Coalition)给出的关于工作流管理系统的定义是：

工作流管理系统是一个软件系统，它通过执行经过计算的流程定义去支持一批专门设定的业务流程。工作流管理系统被用来定义、管理、和执行工作流程。

工作流管理系统的目标：

管理工作的流程以确保工作在正确的时间被期望的人员所执行——在自动化进行的业务过程中插入人工的执行和干预。



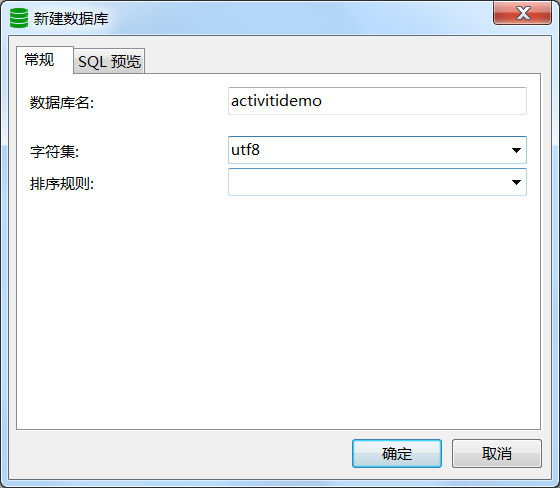
1. Springboot2.X + Activiti5.22集成在线编辑器
2. 环境

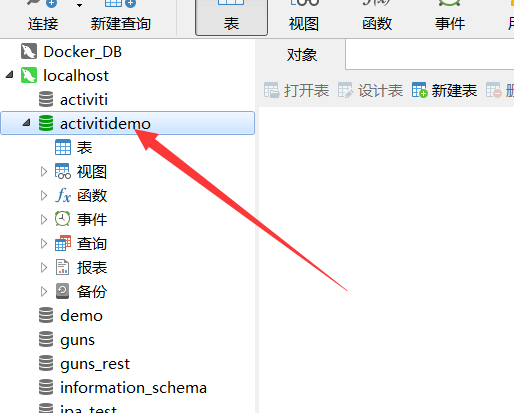
win7 64 + jdk1.8.0\_191 + maven3.6.0 + mysql5.7.24

IDE：IntelliJ IDEA 2018.1.5

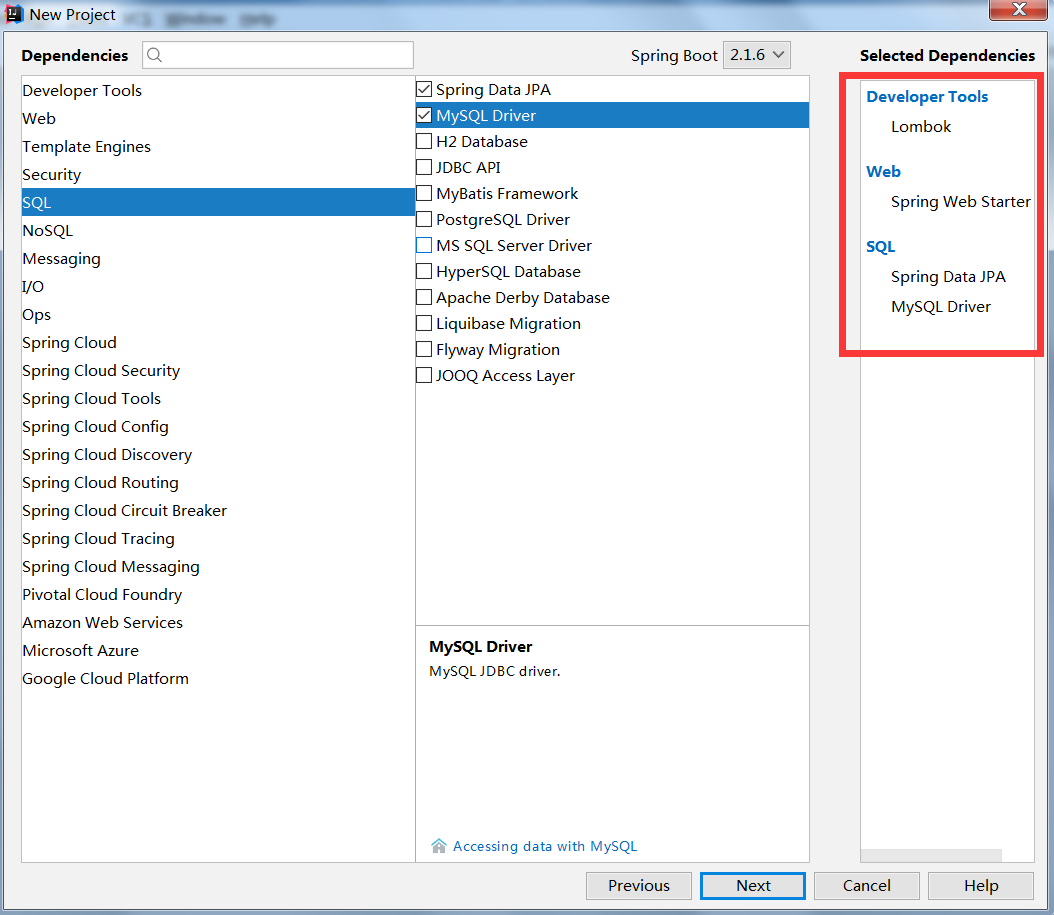
Springboot选择的版本是2.0.6

1. 在本地数据库新建数据库，用于自动生成相关表





1. idea新建springboot工程引入基本础的web包



1. 工程引入activiti，由于我们正常开发中有自己的权限管理，不需要activiti集成的spring security，所以把他移除，最终pom文件为：

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
 <**parent**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-parent</**artifactId**>  
 <**version**>2.0.6.RELEASE</**version**>  
 <**relativePath**/> *<!-- lookup parent from repository -->* </**parent**>  
 <**groupId**>com.showtime</**groupId**>  
 <**artifactId**>app-activitidemo</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 <**name**>app-activitidemo</**name**>  
 <**description**>Demo project for Spring Boot</**description**>  
  
 <**properties**>  
 <**java.version**>1.8</**java.version**>  
 </**properties**>  
  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-data-jpa</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
 <**scope**>runtime</**scope**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.projectlombok</**groupId**>  
 <**artifactId**>lombok</**artifactId**>  
 <**optional**>true</**optional**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-test</**artifactId**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
 *<!--activiti-->* <**dependency**>  
 <**groupId**>org.activiti</**groupId**>  
 <**artifactId**>activiti-spring-boot-starter-basic</**artifactId**>  
 <**version**>5.22.0</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.activiti</**groupId**>  
 <**artifactId**>activiti-spring</**artifactId**>  
 <**version**>5.22.0</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>commons-io</**groupId**>  
 <**artifactId**>commons-io</**artifactId**>  
 <**version**>2.5</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.activiti</**groupId**>  
 <**artifactId**>activiti-modeler</**artifactId**>  
 <**version**>5.22.0</**version**>  
 <**exclusions**>  
 <**exclusion**>  
 <**groupId**>org.springframework.security</**groupId**>  
 <**artifactId**>spring-security-web</**artifactId**>  
 </**exclusion**>  
 <**exclusion**>  
 <**groupId**>org.springframework.security</**groupId**>  
 <**artifactId**>spring-security-config</**artifactId**>  
 </**exclusion**>  
 <**exclusion**>  
 <**groupId**>org.springframework.security</**groupId**>  
 <**artifactId**>spring-security-crypto</**artifactId**>  
 </**exclusion**>  
 </**exclusions**>  
 </**dependency**>  
  
 </**dependencies**>  
  
 <**build**>  
 <**plugins**>  
 <**plugin**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-maven-plugin</**artifactId**>  
 </**plugin**>  
 </**plugins**>  
 </**build**>  
  
</**project**>

1. 修改.yml配置文件，加入配置：

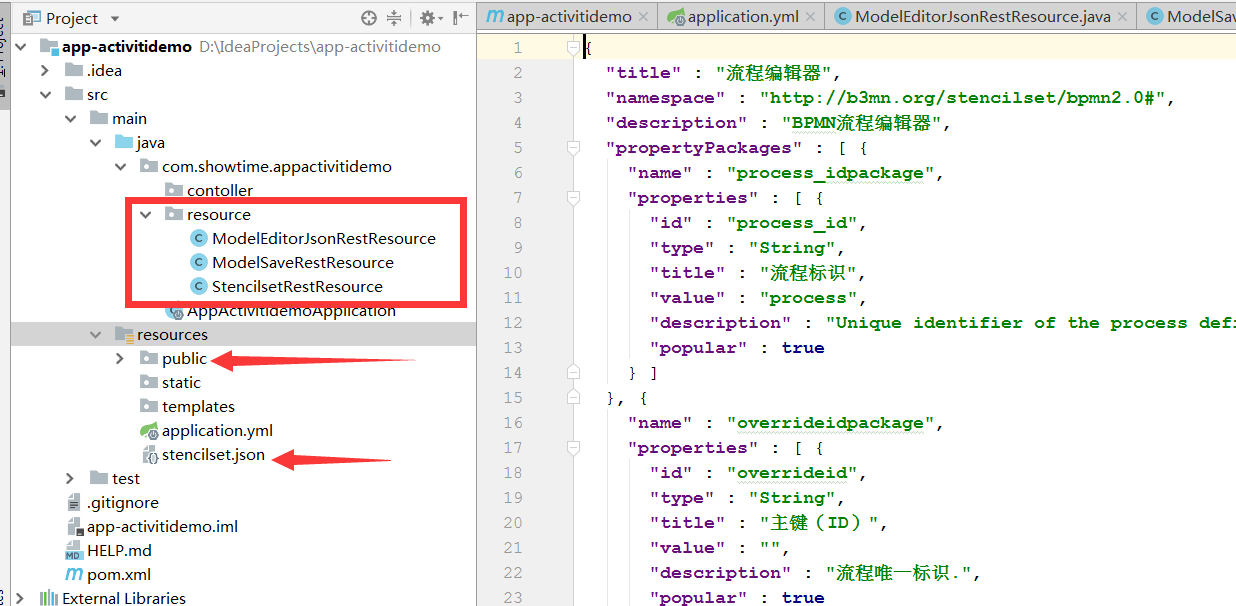
*#整合Activit***spring:  
 jpa:  
 hibernate:  
 ddl-auto:** update *# first：create table ，next :update table* **database:** *MYSQL  
 #整合JPA* **show-sql: true  
 activiti:  
 check-process-definitions: false  
 datasource:  
 url:** jdbc:mysql://127.0.0.1:3306/ activitidemo?characterEncoding=utf-8&useSSL=false  
 **username:** root  
 **password:** 123456  
 **driver-class-name:** com.mysql.jdbc.Driver  
 **maxActive:** 20  
 **initialSize:** 1  
 **maxWait:** 60000  
 **minIdle:** 1  
 **testWhileIdle:** true

1. 引入在线编辑器的相关静态文件和.java文件，这些文件activiti项目的githup可以下载并稍加修改，这里为了后续开发方便我一已经上传到百度云盘链接：

<https://pan.baidu.com/s/1md0Bu7e0_86soKX8HOcDFA> 提取码：0o7l

静态文件直接放到resource目录下、.java文件放入appacctivitidemo包下

终于项目目录结构为：



1. 在controller，先新建类ModelController

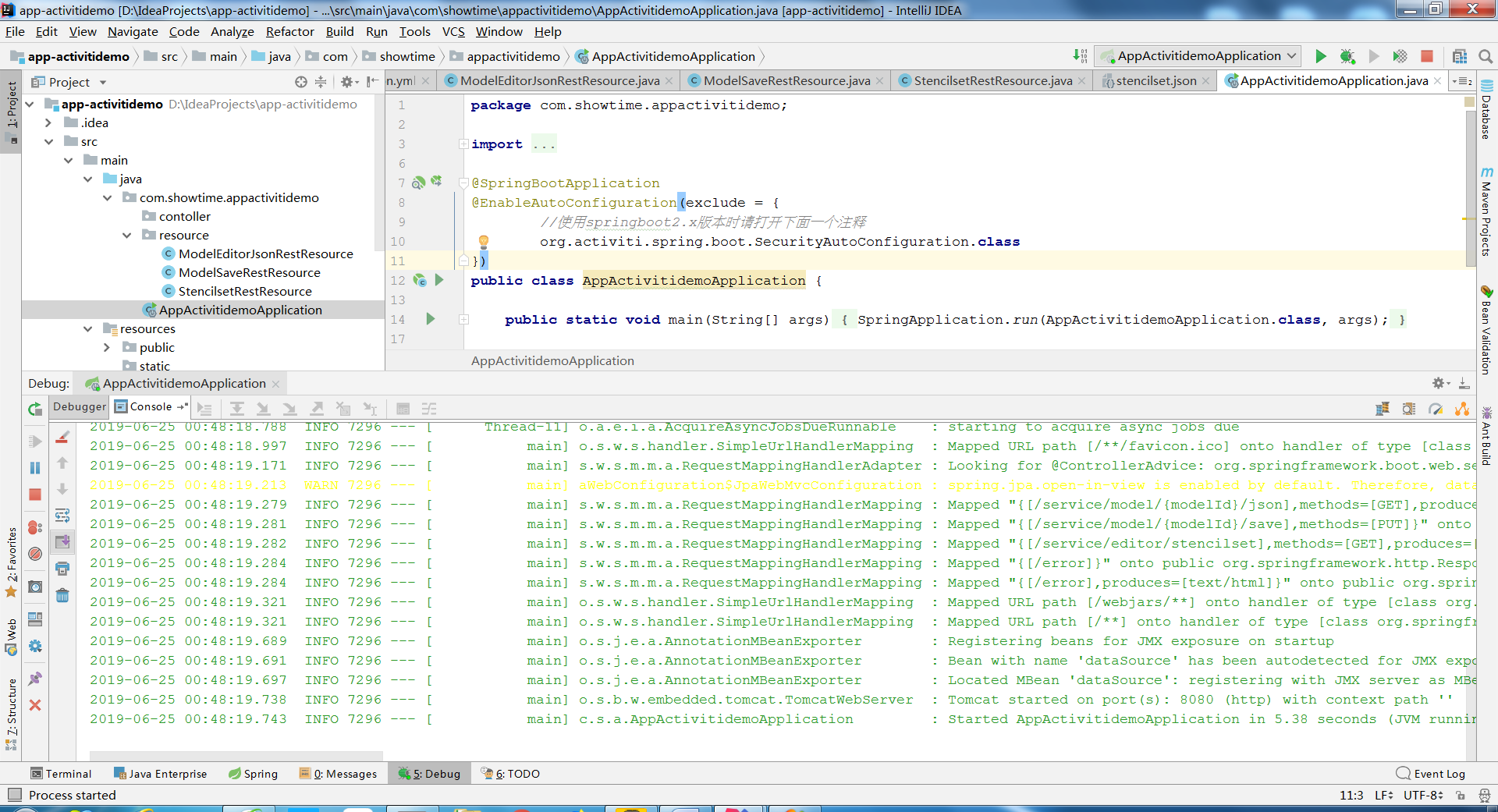
新增方法用于创建新的流程模型：

**package** com.showtime.appactivitidemo.contoller;  
  
**import** com.fasterxml.jackson.databind.ObjectMapper;  
**import** com.fasterxml.jackson.databind.node.ObjectNode;  
**import** lombok.extern.slf4j.Slf4j;  
**import** org.activiti.editor.constants.ModelDataJsonConstants;  
**import** org.activiti.engine.RepositoryService;  
**import** org.activiti.engine.repository.Model;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.web.bind.annotation.GetMapping;  
**import** org.springframework.web.bind.annotation.RequestMapping;  
**import** org.springframework.web.bind.annotation.RestController;  
  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
  
*/\*\*  
 \** ***@Author:*** *Baron  
 \** ***@Description:*** *流程模型controller类  
 \** ***@Date:*** *Created in 2019/6/19 23:24  
 \*/*@RestController  
@Slf4j  
@RequestMapping(**"/model"**)  
**public class** ModelController {  
  
 @Autowired  
 **private** RepositoryService **repositoryService**;  
   
 */\*\*  
 \* 创建流程模型  
 \** ***@param request*** *\** ***@param response*** *\*/* @GetMapping(**"/create"**)  
 **public void** create(HttpServletRequest request, HttpServletResponse response) {  
 **try** {  
 *//设置默认值* String name = **""**;  
 String description = **""**;  
 **int** version = 1;  
 String key = **"1"**;  
 ObjectMapper objectMapper = **new** ObjectMapper();  
 ObjectNode editorNode = objectMapper.createObjectNode();  
 editorNode.put(**"id"**, **"canvas"**);  
 editorNode.put(**"resourceId"**, **"canvas"**);  
 ObjectNode stencilSetNode = objectMapper.createObjectNode();  
 stencilSetNode.put(**"namespace"**, **"http://b3mn.org/stencilset/bpmn2.0#"**);  
 editorNode.put(**"stencilset"**, stencilSetNode);  
 Model modelData = **repositoryService**.newModel();  
  
 ObjectNode modelObjectNode = objectMapper.createObjectNode();  
 modelObjectNode.put(ModelDataJsonConstants.***MODEL\_NAME***, name);  
 modelObjectNode.put(ModelDataJsonConstants.***MODEL\_REVISION***, version);  
 modelObjectNode.put(ModelDataJsonConstants.***MODEL\_DESCRIPTION***, description);  
 modelData.setMetaInfo(modelObjectNode.toString());  
 modelData.setName(name);  
 modelData.setKey(key);  
 *//保存模型* **repositoryService**.saveModel(modelData);  
 **repositoryService**.addModelEditorSource(modelData.getId(), editorNode.toString().getBytes(**"utf-8"**));  
 response.sendRedirect(request.getContextPath() + **"/modeler.html?modelId="** + modelData.getId());  
 } **catch** (Exception e) {  
 ***log***.error(**"流程创建失败，e={}"**,e.getMessage());  
 e.printStackTrace();  
 }  
 }  
  
}

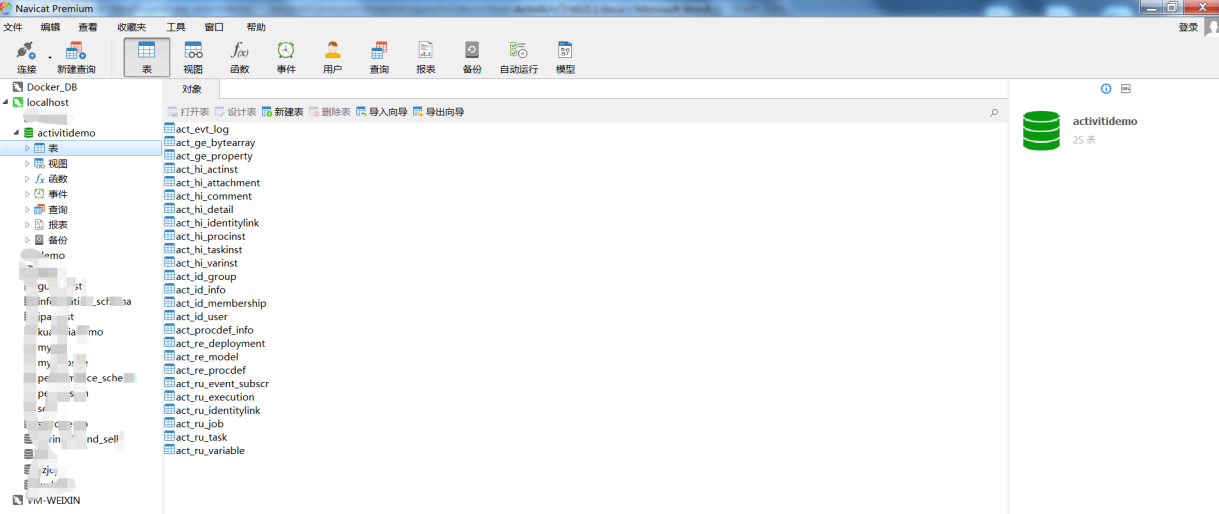


1. 在启动类上加入下面的注解，启动项目：

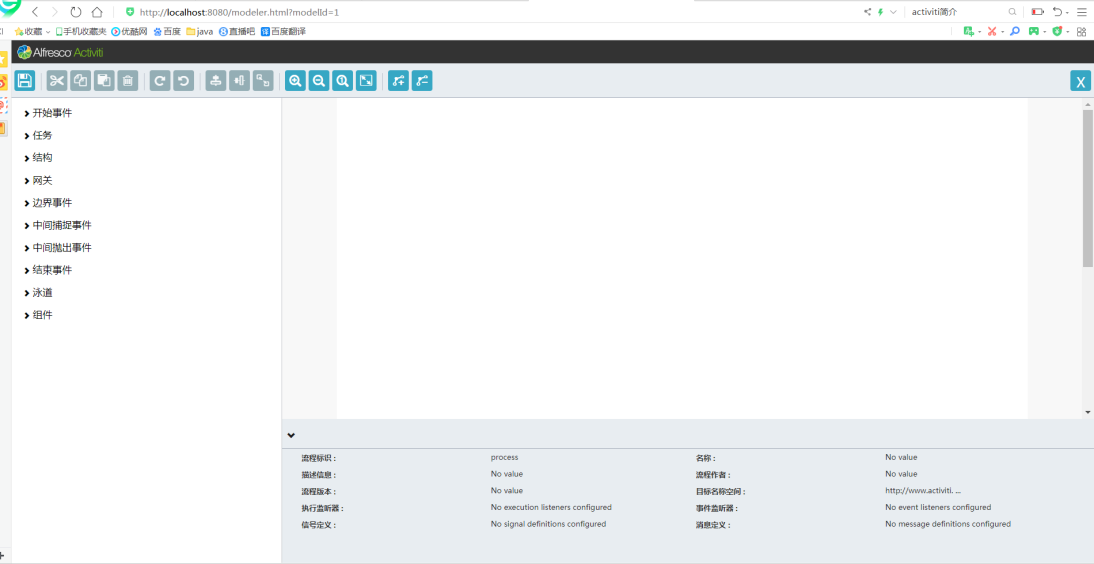
@EnableAutoConfiguration(exclude = {  
 *//使用springboot2.x版本时请打开下面一个注释* org.activiti.spring.boot.SecurityAutoConfiguration.**class**})



1. 查看mysql数据库，可以发现，系统已经自动生成25张表，这些表具体存储的数据，见章末：



1. 浏览器访问url地址http://localhost:8080/model/create，则可打开在线编辑器：



10、Activiti部分表说明：

1：资源库流程规则表

1) act\_re\_deployment 部署信息表

2) act\_re\_model 流程设计模型部署表

3) act\_re\_procdef 流程定义数据表

2：运行时数据库表

1) act\_ru\_execution 运行时流程执行实例表

2) act\_ru\_identitylink 运行时流程人员表，主要存储任务节点与参与者的相关信息

3) act\_ru\_task 运行时任务节点表

4) act\_ru\_variable 运行时流程变量数据表

3：历史数据库表

1) act\_hi\_actinst 历史节点表

2) act\_hi\_attachment 历史附件表

3) act\_hi\_comment 历史意见表

4) act\_hi\_identitylink 历史流程人员表

5) act\_hi\_detail 历史详情表，提供历史变量的查询

6) act\_hi\_procinst 历史流程实例表

7) act\_hi\_taskinst 历史任务实例表

8) act\_hi\_varinst 历史变量表

4：组织机构表

1) act\_id\_group 用户组信息表

2) act\_id\_info 用户扩展信息表

3) act\_id\_membership 用户与用户组对应信息表

4) act\_id\_user 用户信息表

这四张表很常见，基本的组织机构管理，关于用户认证方面建议还是自己开发一套，组件自带的功能太简单，使用中有很多需求难以满足

5：通用数据表

act\_ge\_bytearray 二进制数据表

act\_ge\_property 属性数据表存储整个流程引擎级别的数据,初始化表结构时，会默认插入三条记录

1. 常用service：

工作流七大service简单介绍：

1. repositoryService：流程仓库service,用于管理流程仓库,增删改查流程资源。

2. runtimeService：运行时service,处理正在运行状态的流程实例,任务等。

3. taskService：任务service,管理,查询任务，例如签收，办理,指派任务。

4. historyService：历史service,可以查询所有历史数据,例如,流程实例,任务,活动，变量，附件等。

5. managementService：引擎管理service,和具体业务无关,主要是查询引擎配置，数据库，作业等。

6. identityService：身份service,可以管理和查询用户,组之间的关系。

7. formService：表单service,处理正在运行状态的流程实例,任务等。

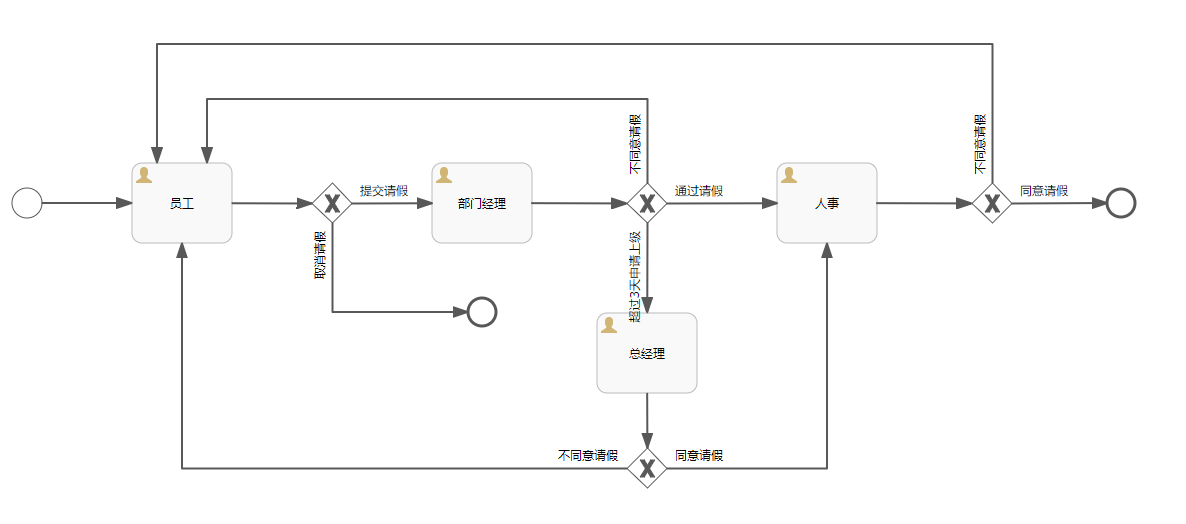
……常用的也就是这几个service，详细用法请百度……

1. 请假流程demo

概述：

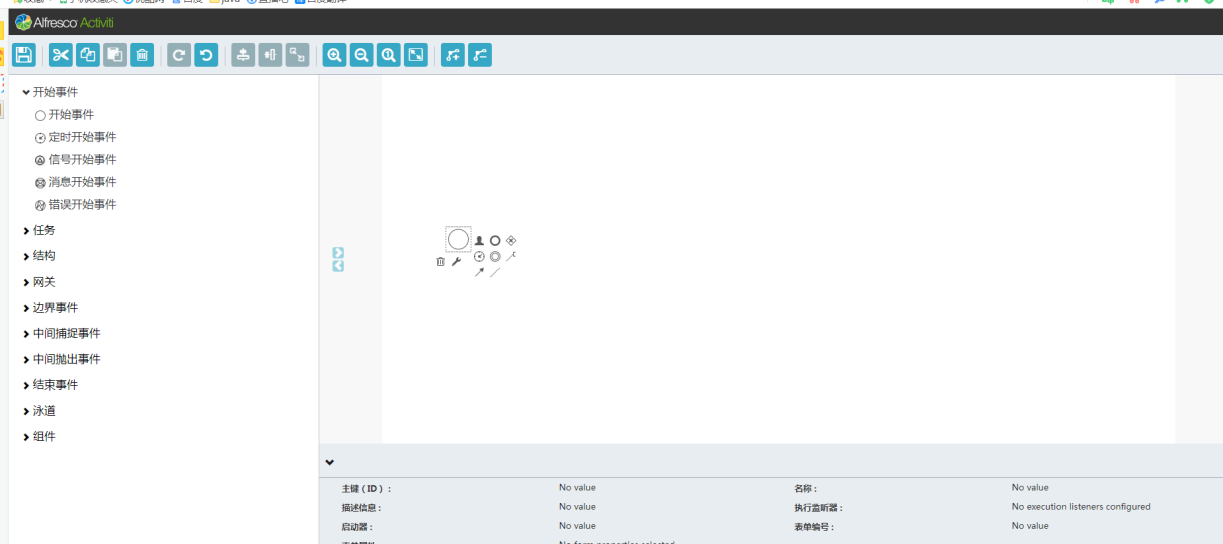
实现员工简易请假流程：

员工向部门经理提出请假要求，部门经理可以通过和退回给员工，部门经理通过后，再到人事审批，人事可以通过和退回给员工，如果员工请假超过3天，部门经理通过后，需要总经理审批通过才能给人事审批，总经理可以通过和退回给员工。

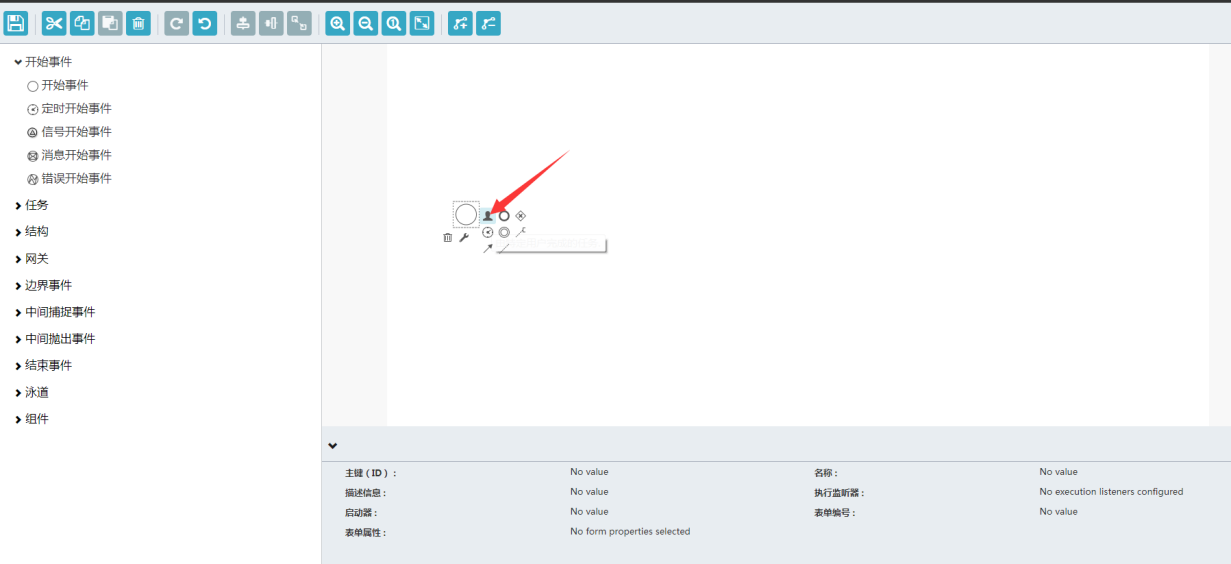


1. 画图部分

1.1 开始：鼠标拖拽开始事件到画图区：



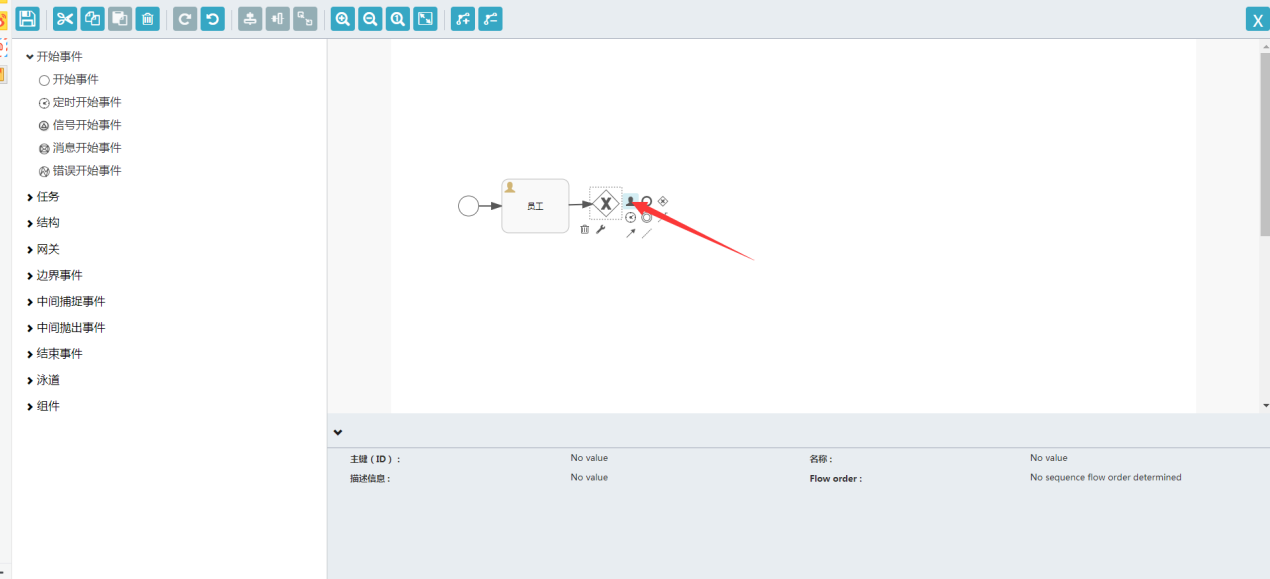
1.2 选择由指定用户完成任务：



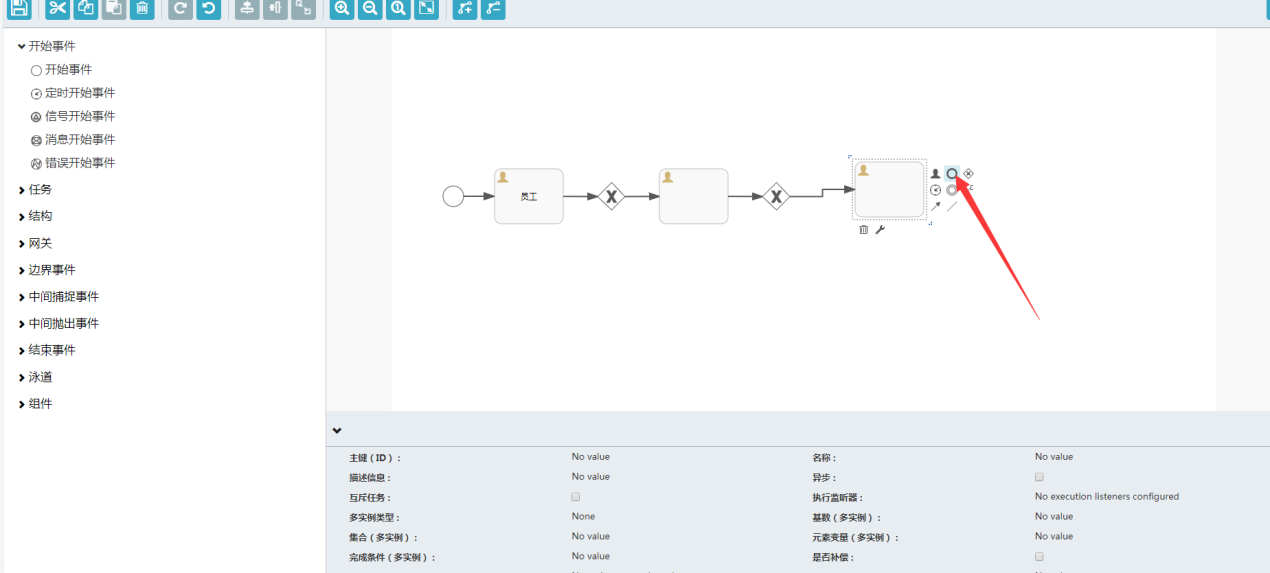
1.3 选择一个网关：



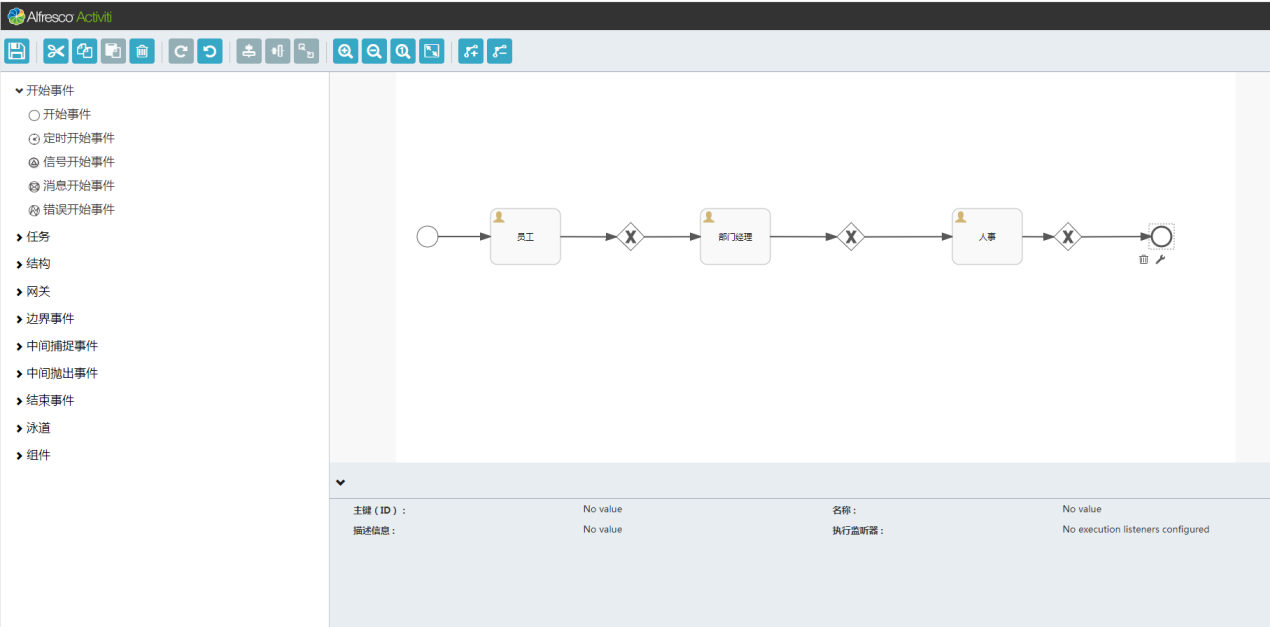
1.4 再选择由指定用户完成任务：



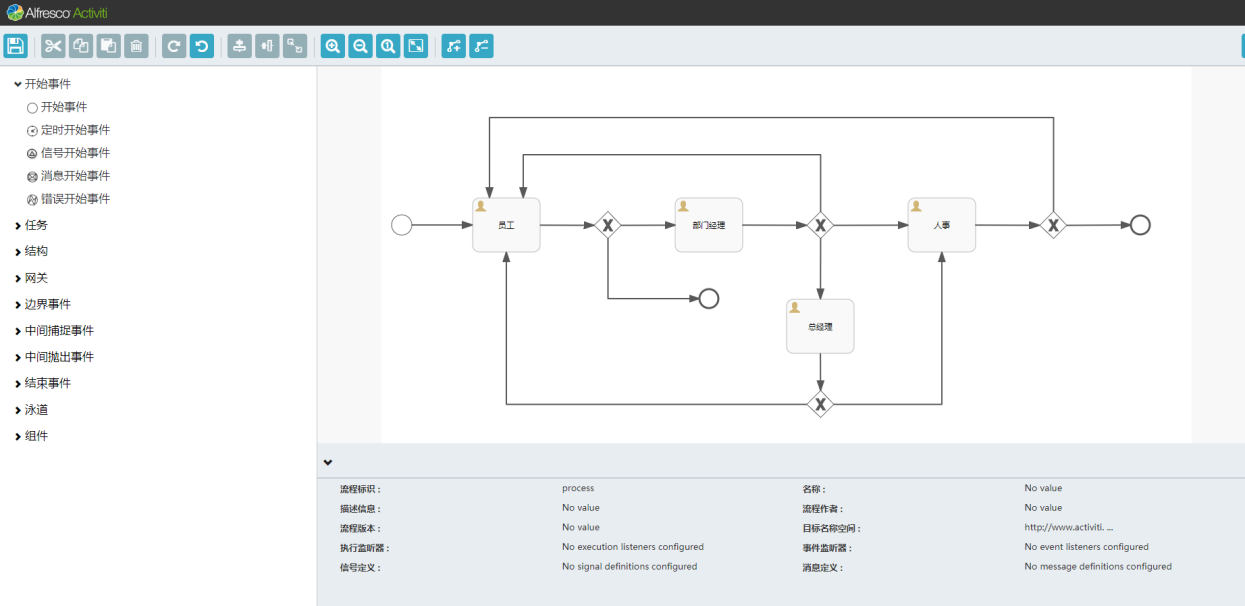
1.5 以次类推直到人事，选择结束：



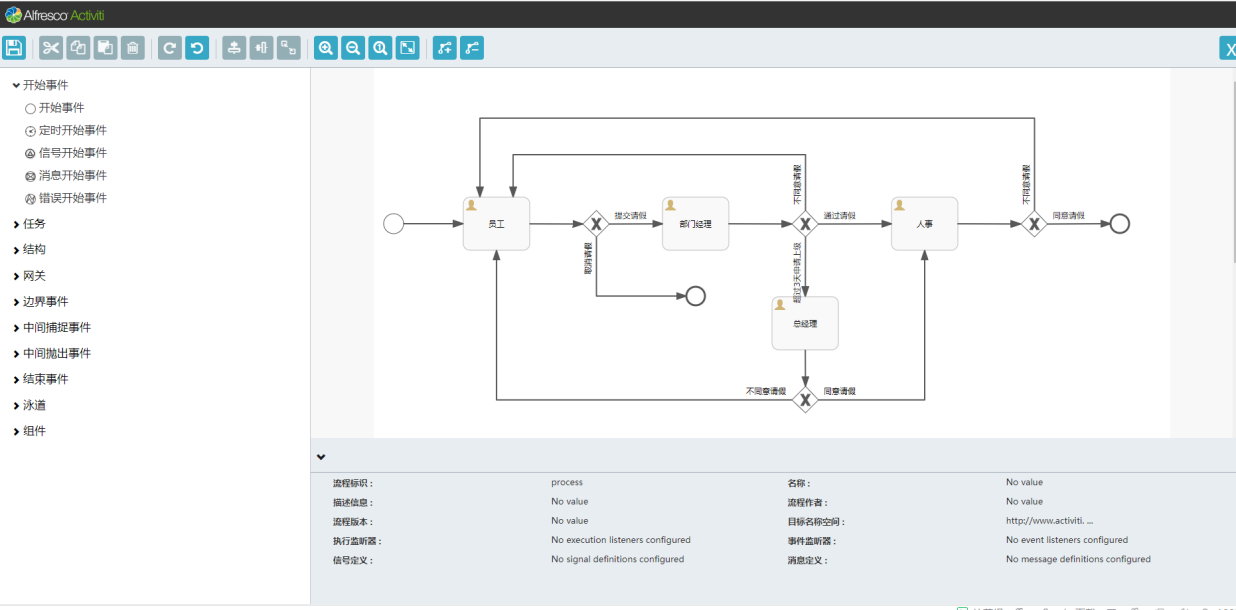
1.6 流程模型完成主线：



1.7 同样通过的拖拽的方法，画出支线：



1.8 补充标识



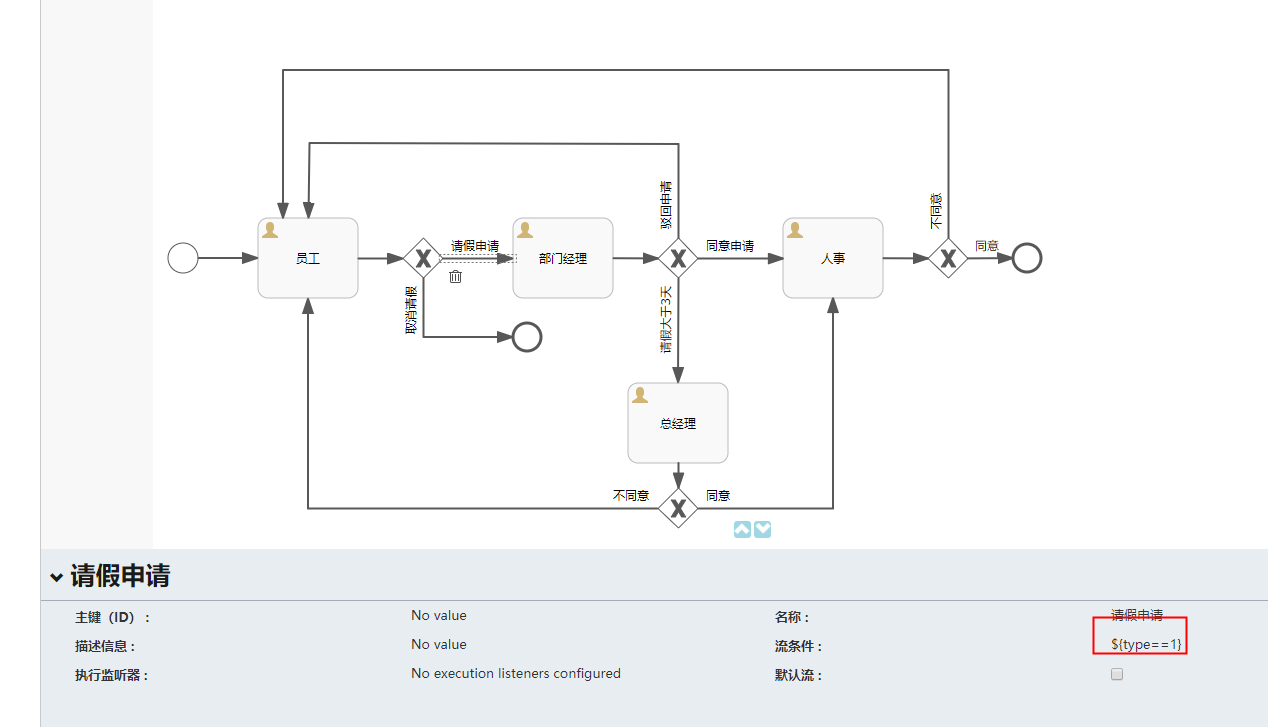
1.9 给网关添加条件，点击网关后的线路，给定条件表达式，规则类似el表达式

0员工取消请假 1员工申请请假到部门经理

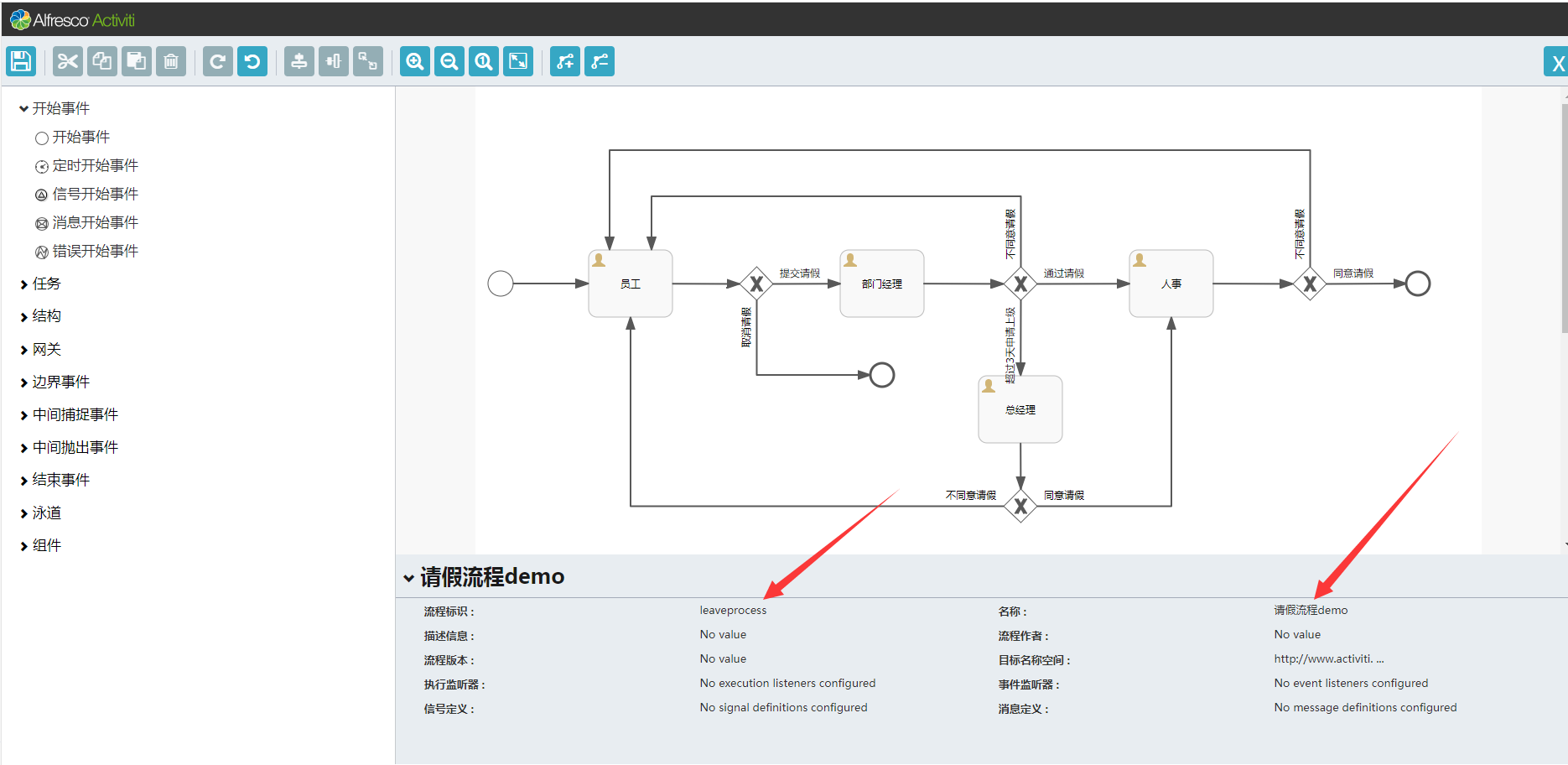
2部门经理驳回 3提交总经理 4总经理驳回 5总经理同意 6部门经理同意

7人事驳回 8人事同意

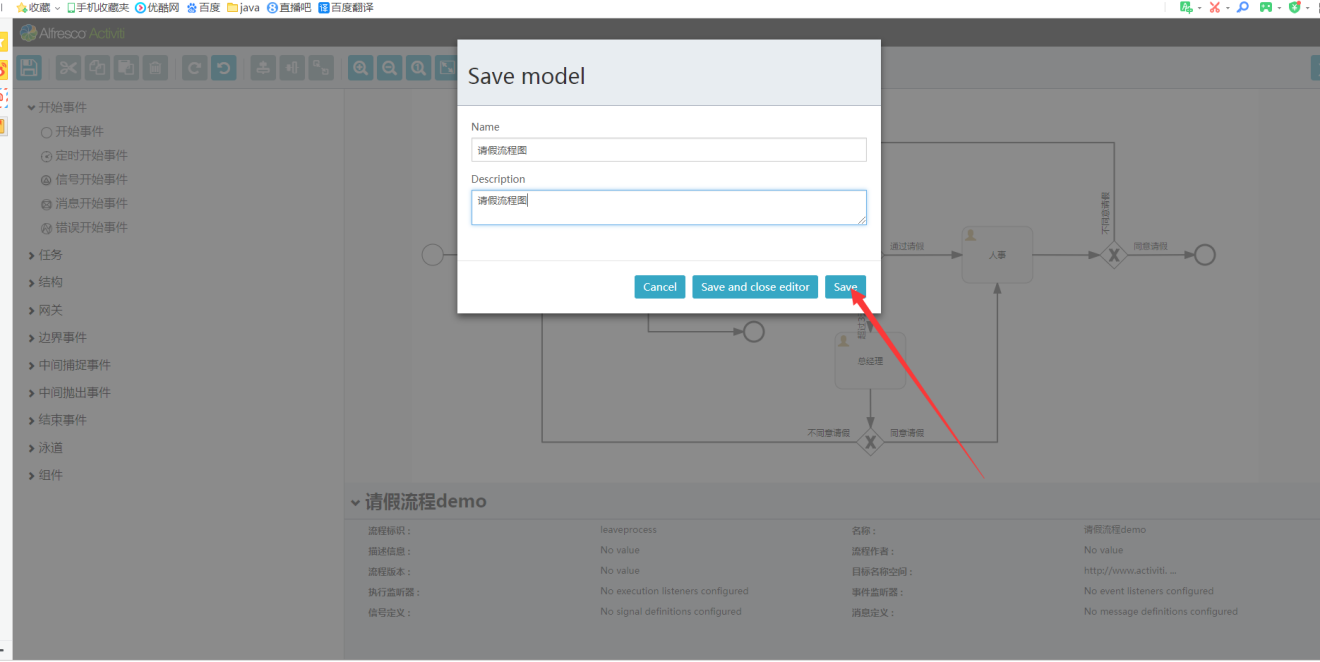
${type==0}



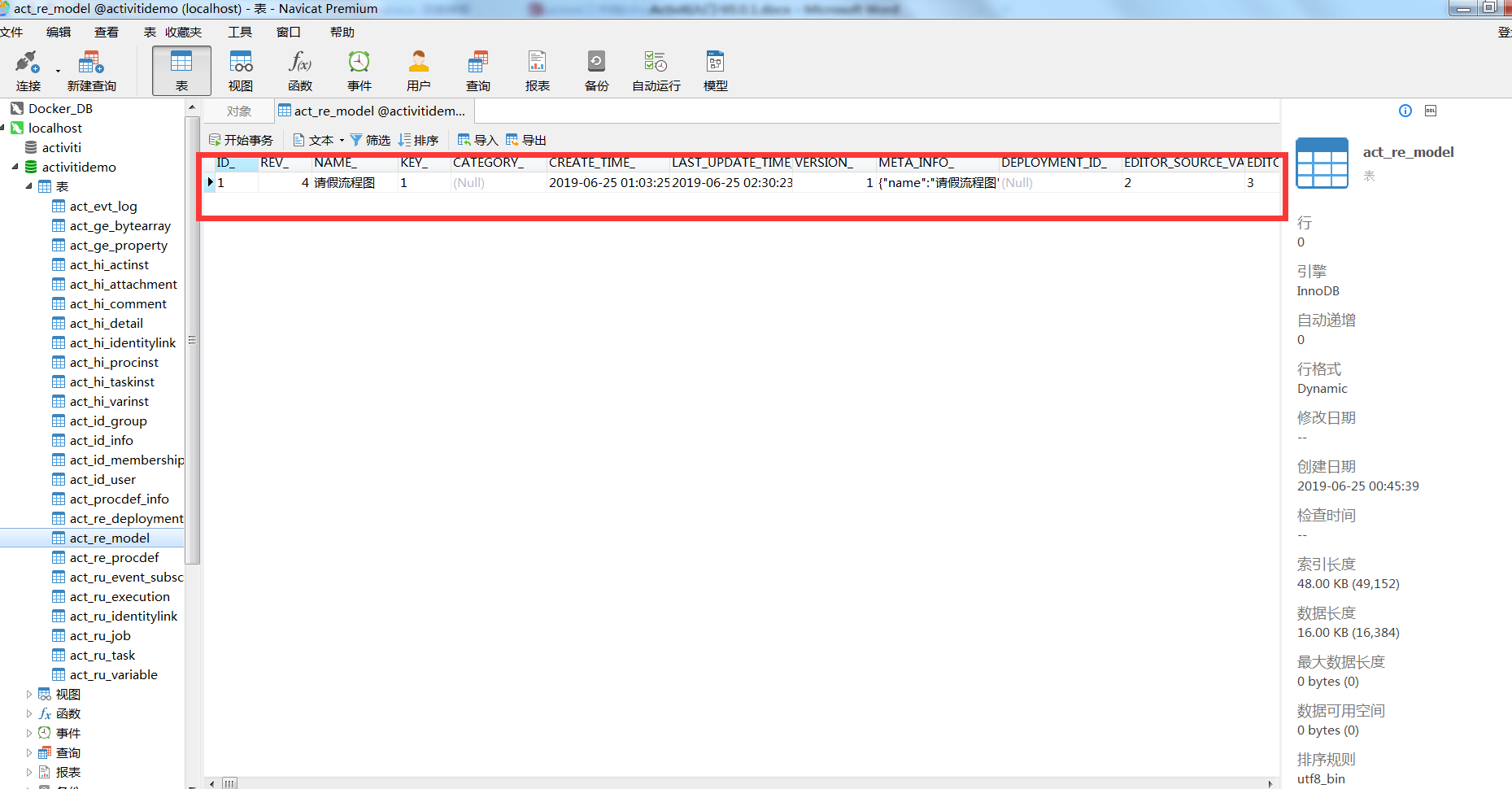
1.10 完成，给定流传标识key和名称，请记住这个key值，后面的流程部署需要用到：



1.11 点击保存按钮，并命名和描述：



1.12 查看数据库的act\_ru\_model表，可以看到刚刚保存的那个画好的模型图



1. 代码部分

2.1 在ModelController添加获取流程模型列表、删除流程模型、流程模型部署方法：具体类如下

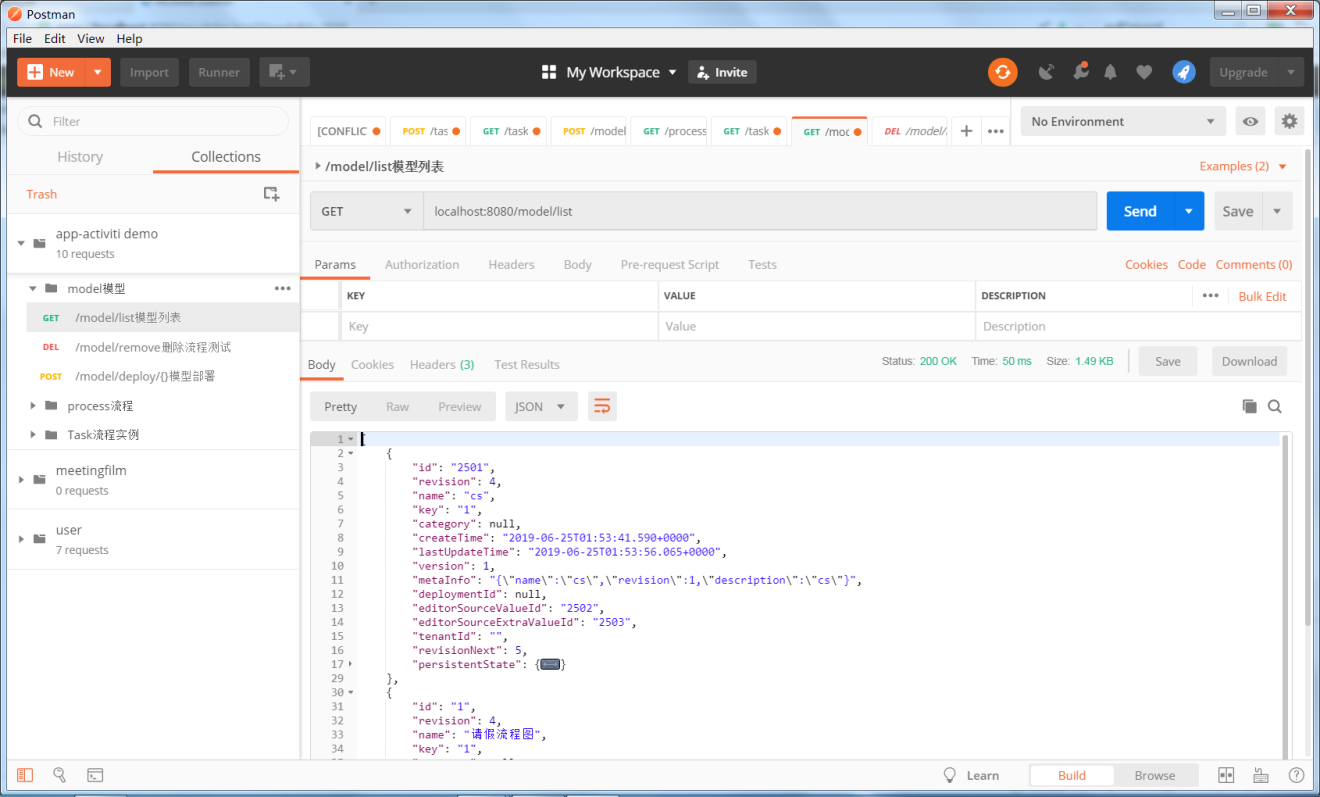
**package** com.cobra.appactiviti.controller;  
  
**import** com.cobra.appactiviti.constant.PageContant;  
**import** com.fasterxml.jackson.databind.JsonNode;  
**import** com.fasterxml.jackson.databind.ObjectMapper;  
**import** com.fasterxml.jackson.databind.node.ObjectNode;  
**import** lombok.extern.slf4j.Slf4j;  
**import** org.activiti.bpmn.converter.BpmnXMLConverter;  
**import** org.activiti.bpmn.model.BpmnModel;  
**import** org.activiti.editor.constants.ModelDataJsonConstants;  
**import** org.activiti.editor.language.json.converter.BpmnJsonConverter;  
**import** org.activiti.engine.RepositoryService;  
**import** org.activiti.engine.repository.Deployment;  
**import** org.activiti.engine.repository.Model;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.http.ResponseEntity;  
**import** org.springframework.web.bind.annotation.\*;  
  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
**import** java.util.HashMap;  
**import** java.util.List;  
**import** java.util.Map;  
  
*/\*\*  
 \** ***@Author:*** *Baron  
 \** ***@Description:*** *流程模型controller类  
 \** ***@Date:*** *Created in 2019/6/19 23:24  
 \*/*@RestController  
@Slf4j  
@RequestMapping(**"/model"**)  
**public class** ModelController {  
  
 @Autowired  
 **private** RepositoryService **repositoryService**;  
  
 */\*\*  
 \* 获取流程模型列表  
 \*  
 \** ***@param page*** *\** ***@param pageNum*** *\** ***@return*** *\*/* @GetMapping(**"/list"**)  
 **public** ResponseEntity getModels(@RequestParam(required = **false**, defaultValue = PageContant.***PAGE***) Integer page,  
 @RequestParam(required = **false**, defaultValue = PageContant.***PAGE\_SIZE***) Integer pageNum) {  
 List<Model> models = **repositoryService**.createModelQuery().orderByCreateTime().desc().listPage(page, pageNum);  
 **return** ResponseEntity.*ok*(models);  
 }  
  
 */\*\*  
 \* 创建流程模型  
 \*  
 \** ***@param request*** *\** ***@param response*** *\*/* @GetMapping(**"/create"**)  
 **public void** create(HttpServletRequest request, HttpServletResponse response) {  
 **try** {  
 *//设置默认值* String name = **""**;  
 String description = **""**;  
 **int** version = 1;  
 String key = **"1"**;  
 ObjectMapper objectMapper = **new** ObjectMapper();  
 ObjectNode editorNode = objectMapper.createObjectNode();  
 editorNode.put(**"id"**, **"canvas"**);  
 editorNode.put(**"resourceId"**, **"canvas"**);  
 ObjectNode stencilSetNode = objectMapper.createObjectNode();  
 stencilSetNode.put(**"namespace"**, **"http://b3mn.org/stencilset/bpmn2.0#"**);  
 editorNode.put(**"stencilset"**, stencilSetNode);  
 Model modelData = **repositoryService**.newModel();  
  
 ObjectNode modelObjectNode = objectMapper.createObjectNode();  
 modelObjectNode.put(ModelDataJsonConstants.***MODEL\_NAME***, name);  
 modelObjectNode.put(ModelDataJsonConstants.***MODEL\_REVISION***, version);  
 modelObjectNode.put(ModelDataJsonConstants.***MODEL\_DESCRIPTION***, description);  
 modelData.setMetaInfo(modelObjectNode.toString());  
 modelData.setName(name);  
 modelData.setKey(key);  
 *//保存模型* **repositoryService**.saveModel(modelData);  
 **repositoryService**.addModelEditorSource(modelData.getId(), editorNode.toString().getBytes(**"utf-8"**));  
 response.sendRedirect(request.getContextPath() + **"/modeler.html?modelId="** + modelData.getId());  
 } **catch** (Exception e) {  
 ***log***.error(**"流程创建失败，e={}"**, e.getMessage());  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* 通过id删除流程模型  
 \*  
 \** ***@param modelId*** *\*/* @DeleteMapping(**"/remove"**)  
 **public** ResponseEntity remove(@RequestParam(**"modelId"**) String modelId) {  
 Map map = **new** HashMap<>(10);  
 **try** {  
 **repositoryService**.deleteModel(modelId);  
 map.put(**"success"**, **false**);  
 map.put(**"msg"**, **"流程删除成功，modelId="** + modelId);  
 } **catch** (Exception e) {  
 map.put(**"success"**, **false**);  
 map.put(**"msg"**, **"流程不存在，modelId="** + modelId);  
 ***log***.error(**"e={}"**, e.getMessage());  
 e.printStackTrace();  
 }  
 **return** ResponseEntity.*ok*(map);  
 }  
  
 */\*\*  
 \* 流程模型部署  
 \*  
 \** ***@param modelId*** *\*/* @PostMapping(**"/deploy/{modelId}"**)  
 **public** ResponseEntity deploy(@PathVariable(**"modelId"**) String modelId) {  
 Map map = **new** HashMap<>(10);  
 Model model = **repositoryService**.getModel(modelId);  
 **if** (model == **null**) {  
 map.put(**"success"**, **false**);  
 map.put(**"msg"**, **"流程不存在，modelId="** + modelId);  
 ***log***.error(**"流程不存在，modelId={}"**, modelId);  
 **return** ResponseEntity.*ok*(map);  
 }  
 **try** {  
 **byte**[] modelByte = **repositoryService**.getModelEditorSource(model.getId());  
 **if** (modelByte == **null**) {  
 map.put(**"success"**, **false**);  
 map.put(**"msg"**, **"模型数据为空,请先设计好，再部署！"**);  
 ***log***.error(**"模型数据为空,请先设计好，再部署！modelId={}"**, modelId);  
 **return** ResponseEntity.*ok*(map);  
 }  
 JsonNode modelNode = **new** ObjectMapper().readTree(modelByte);  
 BpmnModel bpmnModel = **new** BpmnJsonConverter().convertToBpmnModel(modelNode);  
 **if** (bpmnModel.getProcesses().size() == 0) {  
 map.put(**"success"**, **false**);  
 map.put(**"msg"**, **"数据模型不符要求，请至少设计一条主线流程"**);  
 ***log***.error(**"数据模型不符要求，请至少设计一条主线流程"**);  
 **return** ResponseEntity.*ok*(map);  
 }  
 **byte**[] bpmnBytes = **new** BpmnXMLConverter().convertToXML(bpmnModel);  
 *//发布流程* String processName = model.getName() + **".bpmn20.xml"**;  
 Deployment deployment = **repositoryService**.createDeployment()  
 .name(model.getName())  
 .addString(processName, **new** String(bpmnBytes, **"UTF-8"**))  
 .deploy();  
 model.setDeploymentId(deployment.getId());  
 **repositoryService**.saveModel(model);  
 map.put(**"success"**, **true**);  
 map.put(**"msg"**, model.getName() + **"流程发布成功"**);  
 ***log***.info(**"流程发布成功，modelId={}"**, modelId);  
 **return** ResponseEntity.*ok*(map);  
 } **catch** (Exception e) {  
 map.put(**"success"**, **false**);  
 map.put(**"msg"**, model.getName() + **"流程发布失败"**);  
 ***log***.error(**"modelId={}，流程发布失败，e={}"**, modelId, e);  
 }  
 **return** ResponseEntity.*ok*(map);  
 }  
  
}

2.2、添加ProcessController类，获取已经部署的的流程:

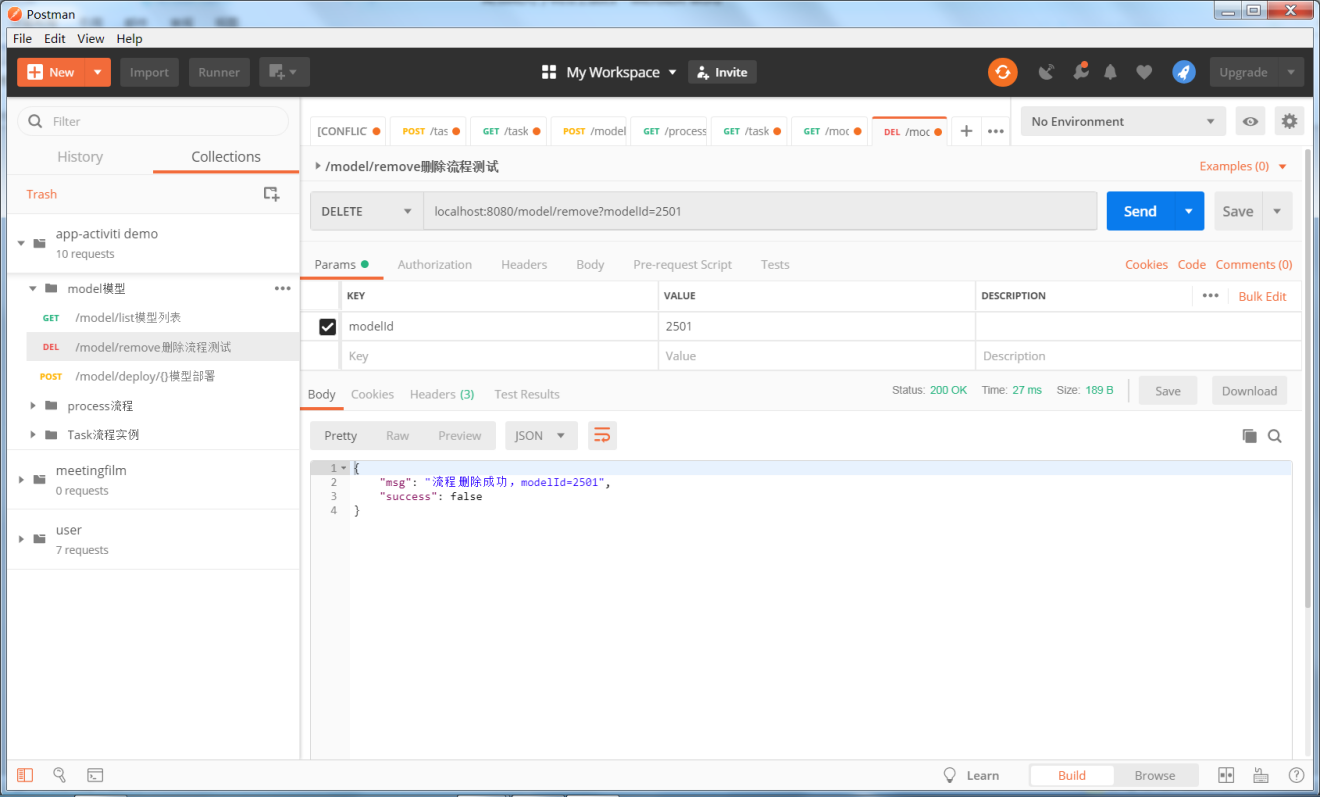
**package** com.showtime.appactivitidemo.contoller;  
  
**import** lombok.extern.slf4j.Slf4j;  
**import** org.activiti.engine.RepositoryService;  
**import** org.activiti.engine.repository.ProcessDefinition;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.http.ResponseEntity;  
**import** org.springframework.web.bind.annotation.GetMapping;  
**import** org.springframework.web.bind.annotation.RequestMapping;  
**import** org.springframework.web.bind.annotation.RestController;  
  
**import** java.util.ArrayList;  
**import** java.util.HashMap;  
**import** java.util.List;  
**import** java.util.Map;  
  
*/\*\*  
 \** ***@Author:*** *Baron  
 \** ***@Description:*** *process流程相关controller  
 \** ***@Date:*** *Created in 2019/6/25 9:45  
 \*/*@RestController  
@RequestMapping(**"/process"**)  
@Slf4j  
**public class** ProcessController {  
  
 @Autowired  
 **private** RepositoryService **repositoryService**;  
  
 */\*\*  
 \*  
 \* 获取已经部署流程  
 \*  
 \** ***@return*** *\*/* @GetMapping(**"/list"**)  
 **public** ResponseEntity getDeployProcess() {  
 List<ProcessDefinition> processDefinitions = **repositoryService**.createProcessDefinitionQuery().  
 orderByProcessDefinitionVersion().desc().listPage(0, 10);  
 List<Map<String, String>> list = **new** ArrayList<>(processDefinitions.size());  
 **for** (ProcessDefinition processDefinition : processDefinitions) {  
 Map map = **new** HashMap<>();  
 map.put(**"id"**,processDefinition.getId());  
 map.put(**"processDefinitionKey"**, processDefinition.getKey());  
 map.put(**"name"**,processDefinition.getName());  
 map.put(**"deploymentId"**,processDefinition.getDeploymentId());  
 map.put(**"description"**,processDefinition.getDescription());  
 list.add(map);  
 }  
 **return** ResponseEntity.*ok*(list);  
 }  
  
}

2.3 对2.1和2.2的测试:

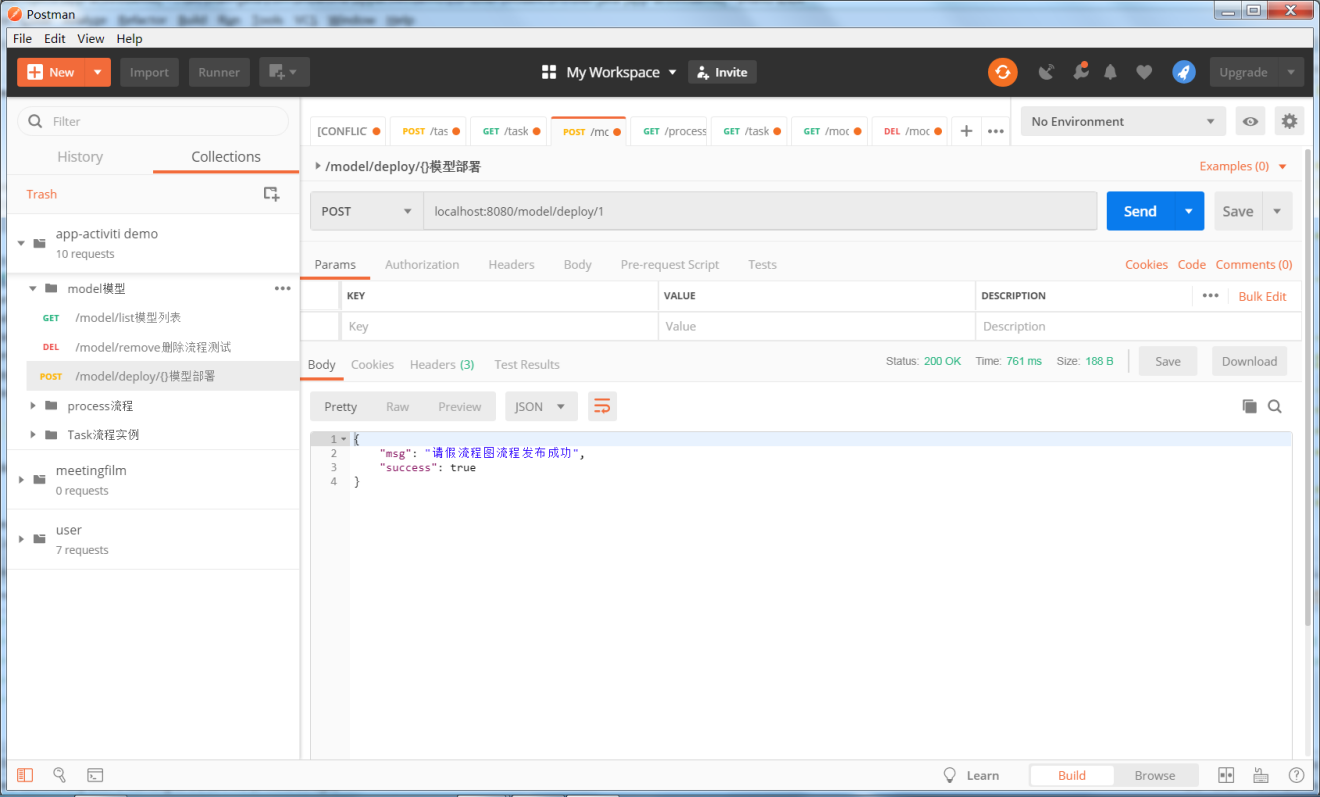
获取模型列表localhost:8080/model/list：



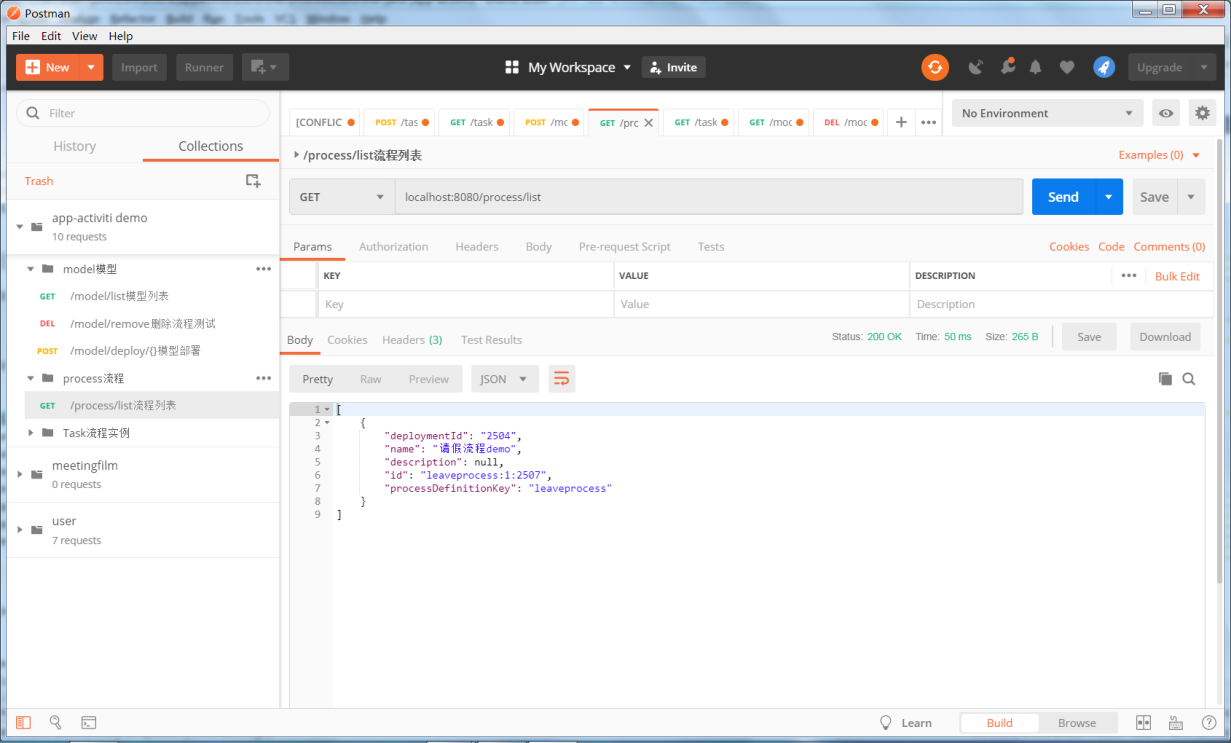
删除模型localhost:8080/model/remove?modelId=2501：



部署流程模型localhost:8080/model/deploy/1：



获取已部署流程列表：localhost:8080/process/list

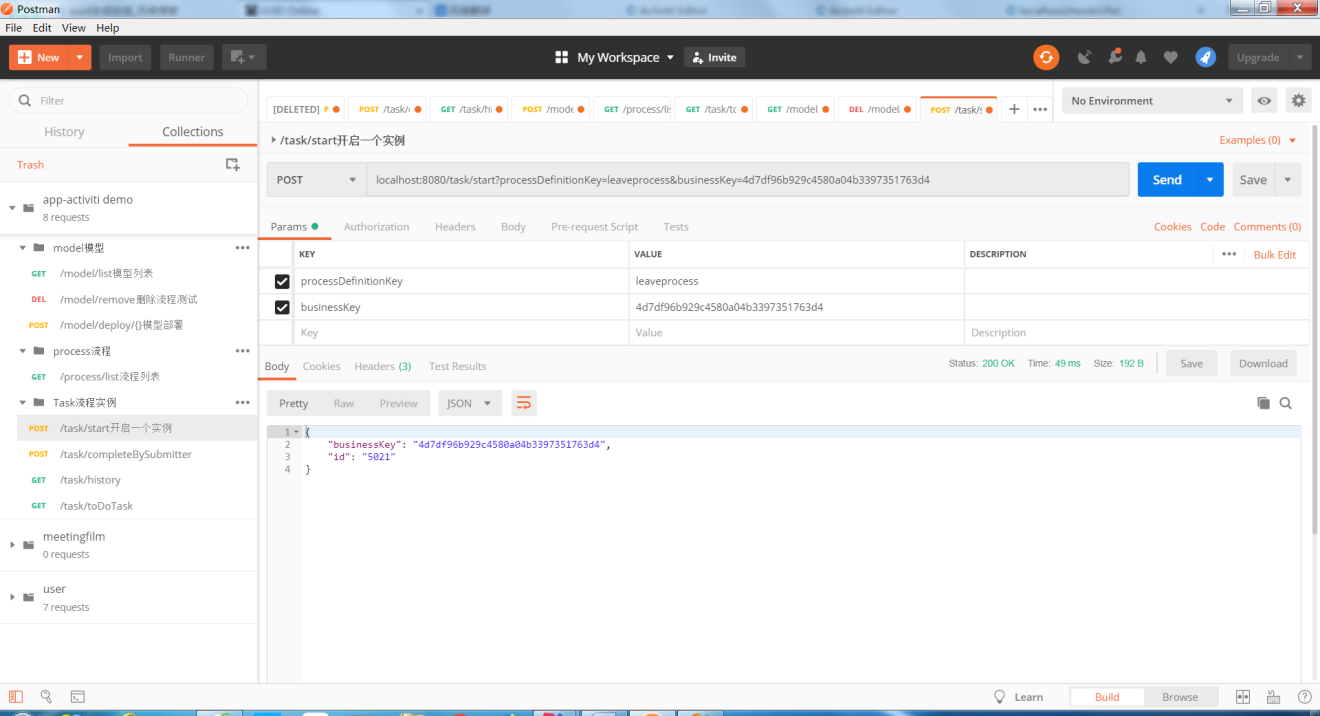


2.4 新增任务管理controller类，添加开启流程实例的、提交人完成本人环节任务、根据委办人查询待办任务、委办人完成自己的任务、根据请假事件id查询请假事件历史环节表5个方法，具体如下：

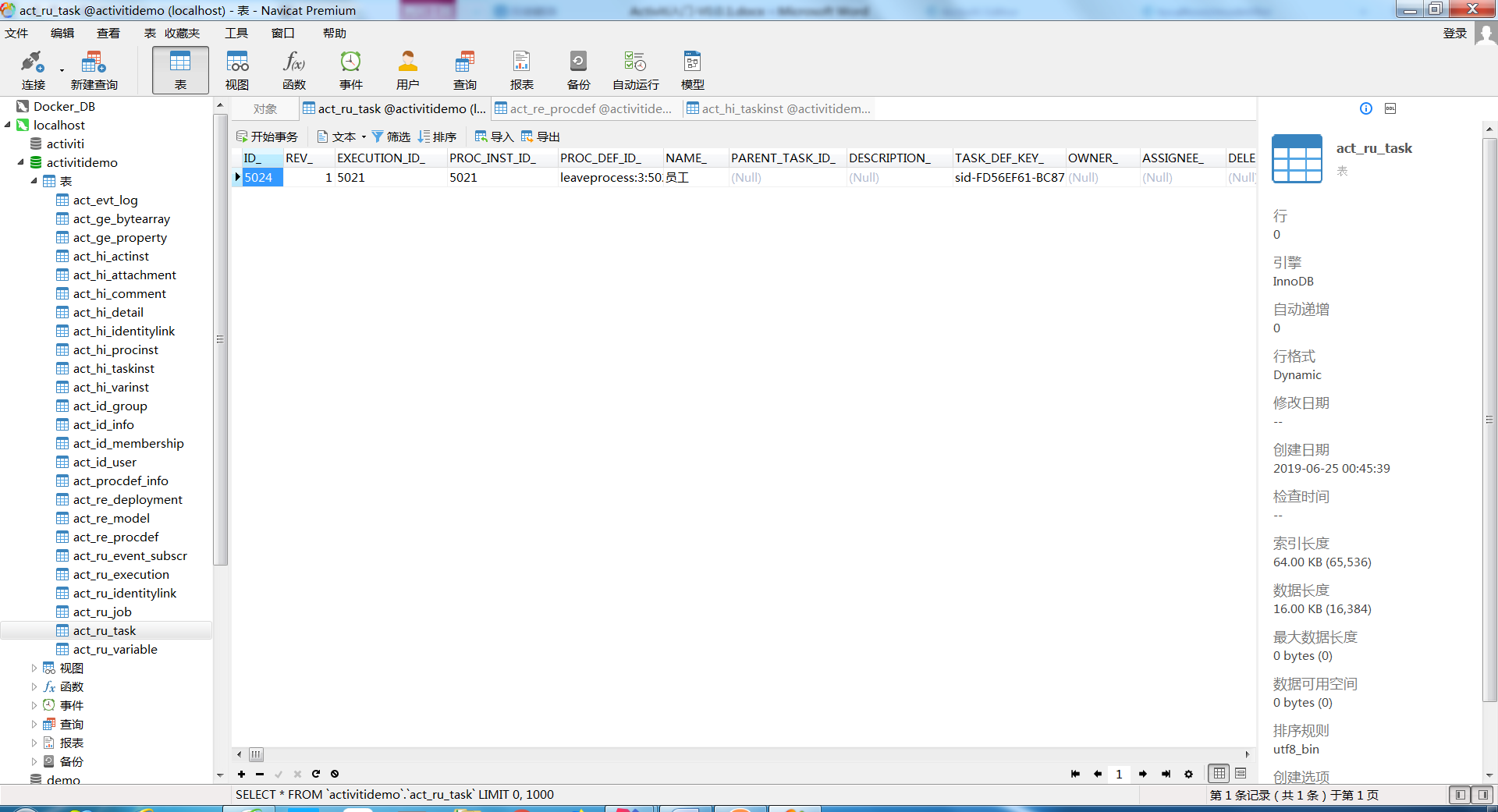
**package** com.showtime.appactivitidemo.contoller;  
  
**import** lombok.extern.slf4j.Slf4j;  
**import** org.activiti.engine.HistoryService;  
**import** org.activiti.engine.RuntimeService;  
**import** org.activiti.engine.TaskService;  
**import** org.activiti.engine.history.HistoricTaskInstance;  
**import** org.activiti.engine.runtime.ProcessInstance;  
**import** org.activiti.engine.task.Task;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.http.ResponseEntity;  
**import** org.springframework.web.bind.annotation.\*;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.Map;  
**import** java.util.concurrent.ConcurrentHashMap;  
  
*/\*\*  
 \** ***@Author:*** *Baron  
 \** ***@Description:*** *task流程实例controller  
 \** ***@Date:*** *Created in 2019/6/25 10:03  
 \*/*@RestController  
@RequestMapping(**"/task"**)  
@Slf4j  
**public class** TaskController {  
  
 @Autowired  
 **private** RuntimeService **runtimeService**;  
  
 @Autowired  
 **private** TaskService **taskService**;  
  
 @Autowired  
 **private** HistoryService **historyService**;  
  
  
 */\*\*  
 \* 开启流程实例  
 \** ***@param processDefinitionKey*** *流程key  
 \** ***@param businessKey*** *事件id，这里用的是记录请假事件的id  
 \** ***@return*** *\*/* @PostMapping(**"/start"**)  
 **public** ResponseEntity startTask(@RequestParam(**"processDefinitionKey"**) String processDefinitionKey,  
 @RequestParam(**"businessKey"**) String businessKey) {  
 Map<String, String> result = **new** ConcurrentHashMap<>(16);  
 ProcessInstance processInstance = **runtimeService**.startProcessInstanceByKey(processDefinitionKey, businessKey);  
 result.put(**"id"**, processInstance.getId());  
 result.put(**"businessKey"**, processInstance.getBusinessKey());  
 ***log***.info(**"processInstance={}"**, processInstance);  
 ProcessInstance result1 = **runtimeService**.createProcessInstanceQuery().processInstanceId(processInstance.getId()).singleResult();  
 System.***out***.println(result1.equals(processInstance));  
 **return** ResponseEntity.*ok*(result);  
 }  
  
 */\*\*  
 \* 员工完成请假提交的接口  
 \** ***@param businessKey*** *请假事件id  
 \** ***@param assignee*** *委办人/向谁请假  
 \** ***@param condition*** *提交/取消（1为提交，0为取消）  
 \** ***@return*** *\*/* @PostMapping(**"/completeBySubmitter"**)  
 **public** ResponseEntity completeTaskWithOrder(@RequestParam(**"businessKey"**) String businessKey,  
 @RequestParam(**"assignee"**) String assignee,  
 @RequestParam(**"condition"**) Integer condition) {  
 Map<String, String> result = **new** ConcurrentHashMap<>(16);  
 Task task = **taskService**.createTaskQuery().  
 processInstanceBusinessKeyLike(businessKey).singleResult();  
 ***log***.info(**"task={}"**, task);  
 result.put(**"taskId"**, task.getId());  
 *//设置流转条件* Map<String, Object> variables = **new** ConcurrentHashMap<>(16);  
 variables.put(**"type"**,condition);  
 **taskService**.complete(task.getId(),variables);  
 Task task1 = **taskService**.createTaskQuery().  
 processInstanceBusinessKeyLike(businessKey).singleResult();  
 **taskService**.setAssignee(task1.getId(), assignee);  
 **return** ResponseEntity.*ok*(result);  
 }  
  
  
 */\*\*  
 \* 根据指定委办人，查询代办事件  
 \*  
 \** ***@param assignee*** *委办人的标识  
 \** ***@return*** *\*/* @GetMapping(**"/toDoTasksByAssignee"**)  
 **public** ResponseEntity getToDoTaskList(String assignee) {  
 List<Map<String, Object>> list = **null**;  
 *//指定人* List<Task> listByAssignee = **taskService**.createTaskQuery().taskAssignee(assignee).list();  
 **if** (listByAssignee != **null** && listByAssignee.size() > 0) {  
 list = **new** ArrayList<>(listByAssignee.size());  
 **for** (Task task : listByAssignee) {  
 Map<String, Object> map = **new** ConcurrentHashMap<>();  
 map.put(**"taskAssignee"**,task.getAssignee());  
 map.put(**"taskId"**,task.getId());  
 map.put(**"taskProcessInstanceId"**,task.getProcessInstanceId());  
 list.add(map);  
 }  
 }  
 **return** ResponseEntity.*ok*(list);  
 }  
  
 */\*\*  
 \* 委办人完成自己流转到自己这一环节的任务  
 \** ***@param businessKey*** *业务id  
 \** ***@param handler*** *处理人  
 \** ***@param assignee*** *下一个委办人  
 \** ***@param condition*** *流传条件  
 \** ***@return*** *\*/* @PostMapping(**"/completeByAssignee"**)  
 **public** ResponseEntity completeTaskByAssigneeWithOrder(@RequestParam(**"businessKey"**) String businessKey,  
 @RequestParam(**"handler"**) String handler,  
 @RequestParam(**"assignee"**) String assignee,  
 @RequestParam(**"condition"**) Integer condition) {  
 Map<String, String> result = **new** ConcurrentHashMap<>(16);  
 Task task = **taskService**.createTaskQuery().  
 processInstanceBusinessKeyLike(businessKey).singleResult();  
 ***log***.info(**"task={}"**, task);  
 **if** (!handler.equals(task.getAssignee())) {  
 result.put(**"msg"**,**"执行失败，不是指定委办人没有权限"**);  
 result.put(**"task.getAssignee()"**,task.getAssignee());  
 **return** ResponseEntity.*ok*(result);  
 }  
 result.put(**"taskId"**, task.getId());  
 *//设置流转条件* Map<String, Object> variables = **new** ConcurrentHashMap<>(16);  
 variables.put(**"type"**,condition);  
 **taskService**.complete(task.getId(),variables);  
 Task task1 = **taskService**.createTaskQuery().  
 processInstanceBusinessKeyLike(businessKey).singleResult();  
 *//如果流程还没完成，设置下一个委办人* **if** (task1 != **null**) {  
 **taskService**.setAssignee(task1.getId(), assignee);  
 }  
 **return** ResponseEntity.*ok*(result);  
 }  
  
  
 */\*\*  
 \* 根据businessKey获取操作信息  
 \*  
 \** ***@param businessKey*** *业务id  
 \** ***@return*** *\*/* @GetMapping(**"/processInstanceTaskHistory"**)  
 **public** ResponseEntity getHistory(@RequestParam(**"businessKey"**) String businessKey) {  
 List<HistoricTaskInstance> list = **historyService**.createHistoricTaskInstanceQuery().  
 processInstanceBusinessKey(businessKey).list();  
 **for** (HistoricTaskInstance historicTaskInstance : list) {  
 *//获取task评论和附件* System.***out***.println(historicTaskInstance.getStartTime());  
 **taskService**.getComment(historicTaskInstance.getId());  
 **taskService**.getAttachment(historicTaskInstance.getId());  
 }  
 ***log***.info(**"list={}"**, list);  
 **return** ResponseEntity.*ok*(list);  
 }  
  
  
  
}

2.5 测试：走一个请假流程

2.5.1 开启一个请假流程实例：

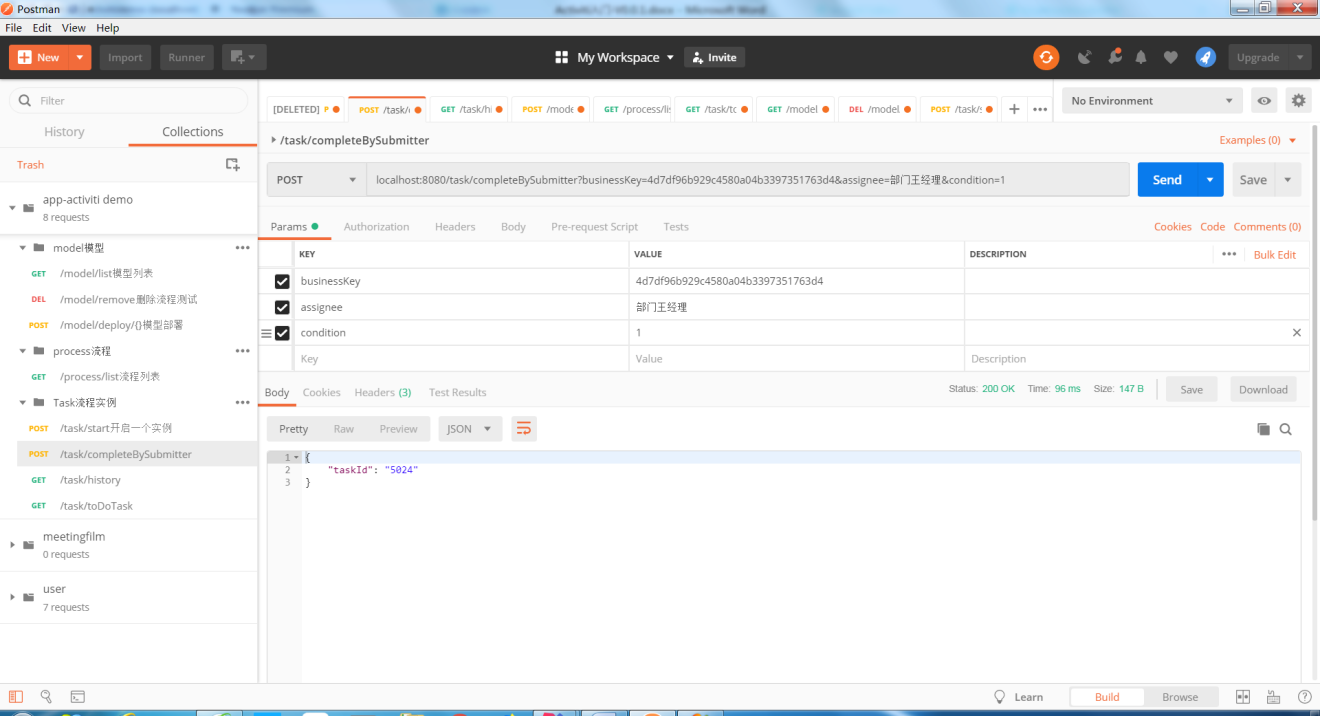
localhost:8080/task/start?processDefinitionKey=leaveprocess&businessKey=4d7df96b929c4580a04b3397351763d4

数据库多出一条数据

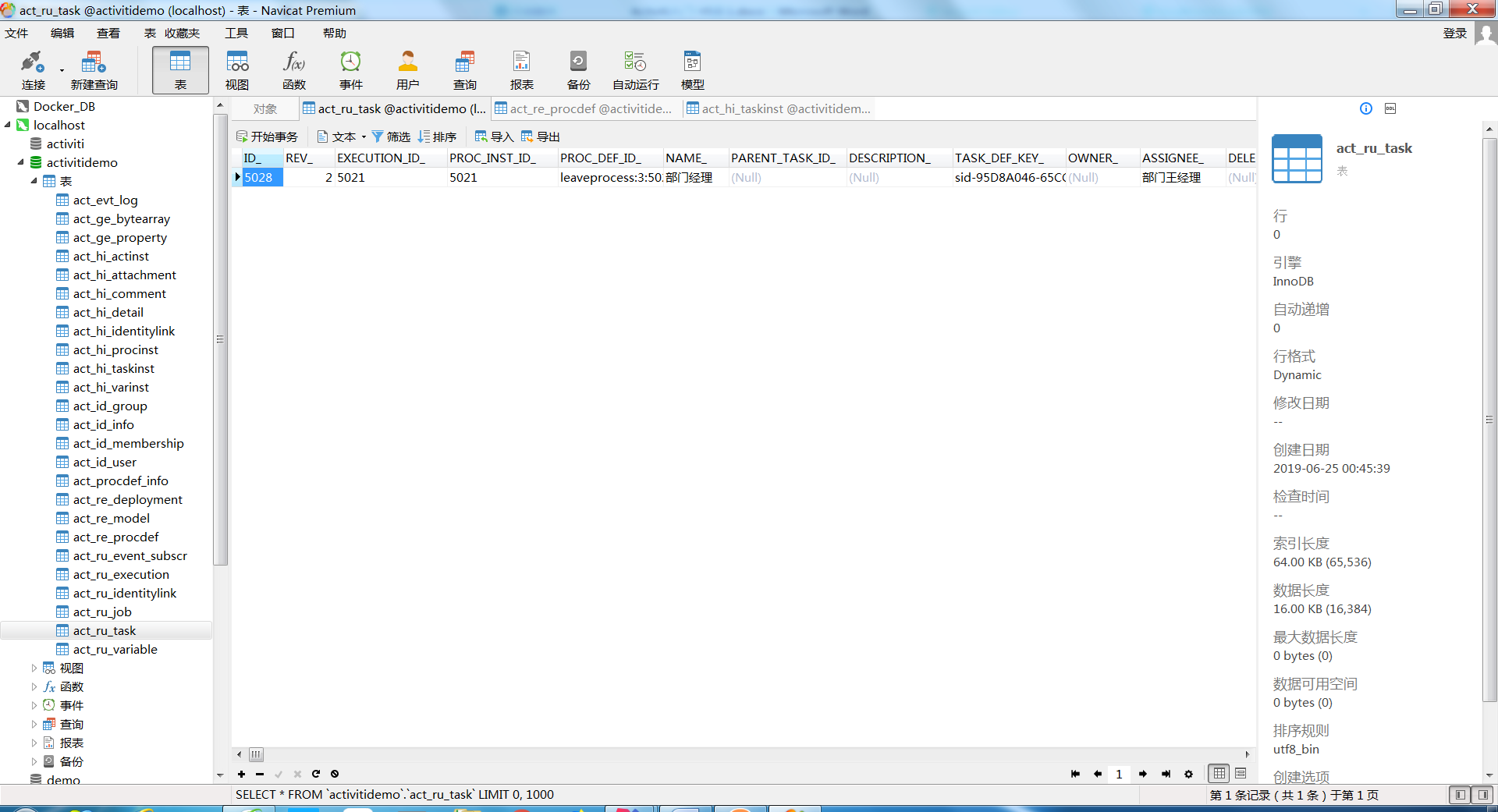


2.5.2 员工给定代办人（部门王经理），完成申请

localhost:8080/task/completeBySubmitter?businessKey=eb807a4602844f7b9dcc6eb635cb77e2&assignee=部门王经理&condition=1

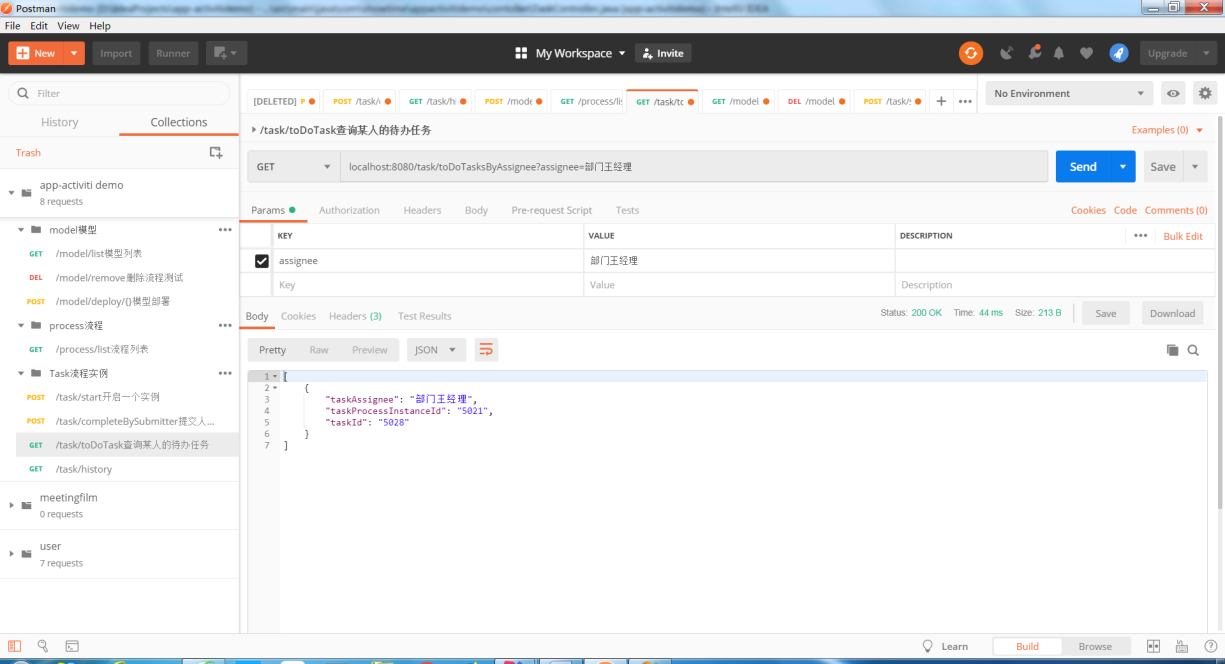


数据库数据有变动



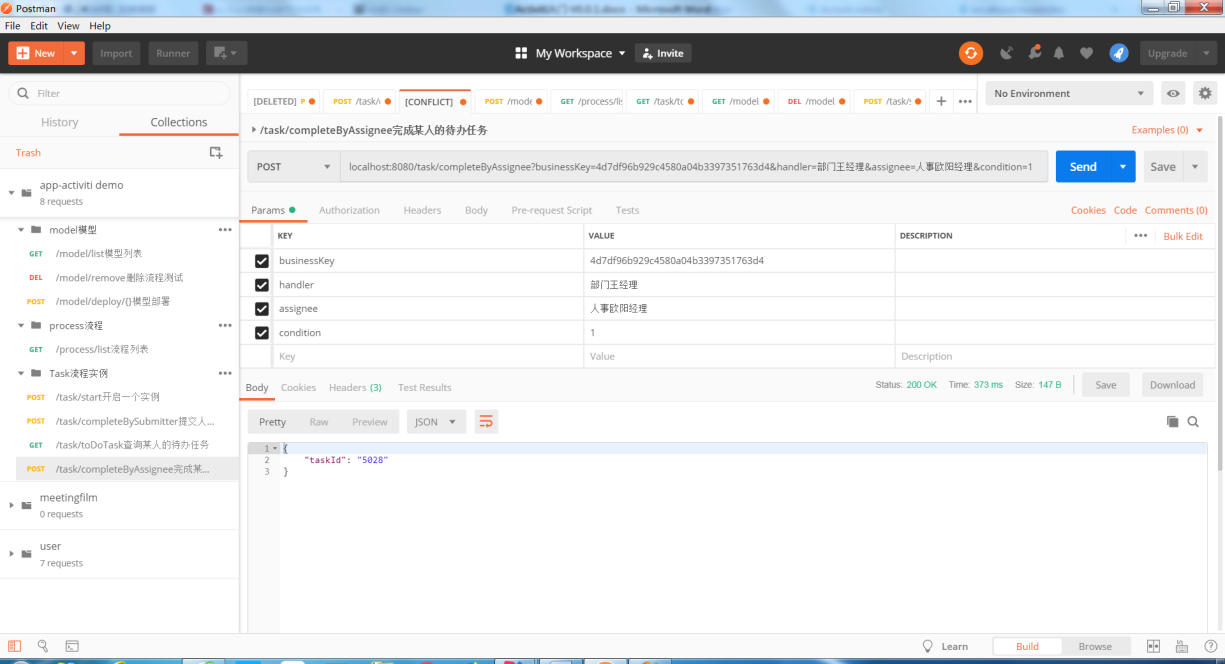
2.5.3 部门王经理查询待办任务：

localhost:8080/task/toDoTasksByAssignee?assignee=部门王经理

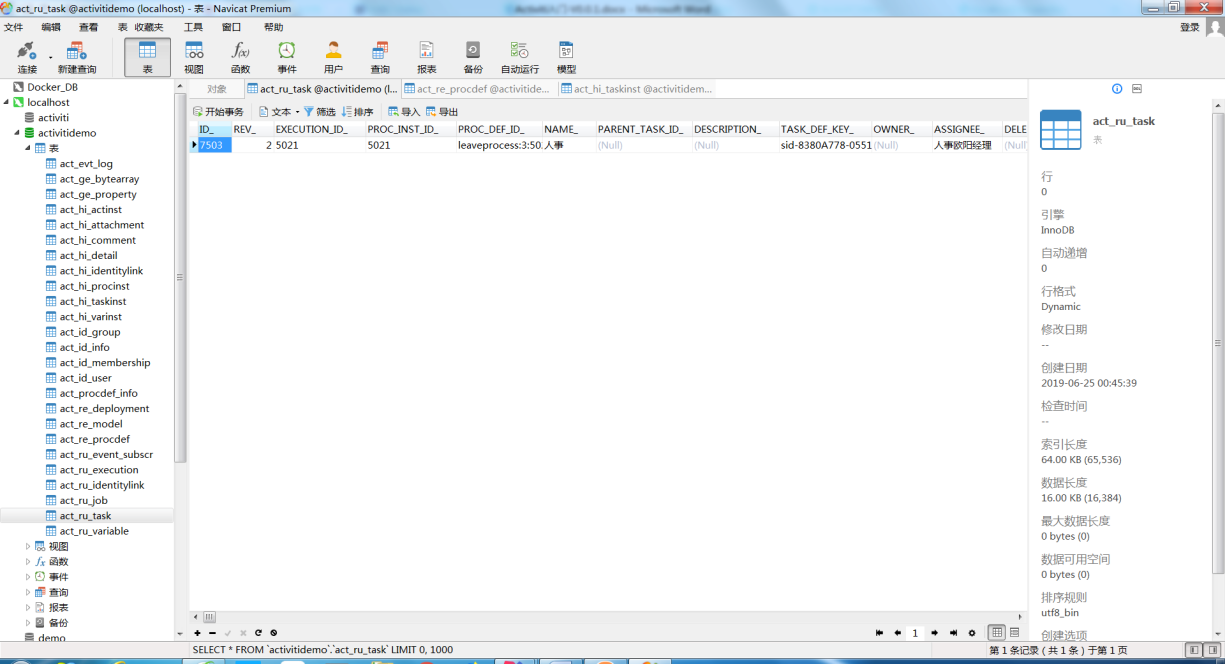


2.5.4 部门王经理通过审批，任务流到人事经理：

localhost:8080/task/completeByAssignee?businessKey=4d7df96b929c4580a04b3397351763d4&handler=部门王经理&assignee=人事欧阳经理&condition=1

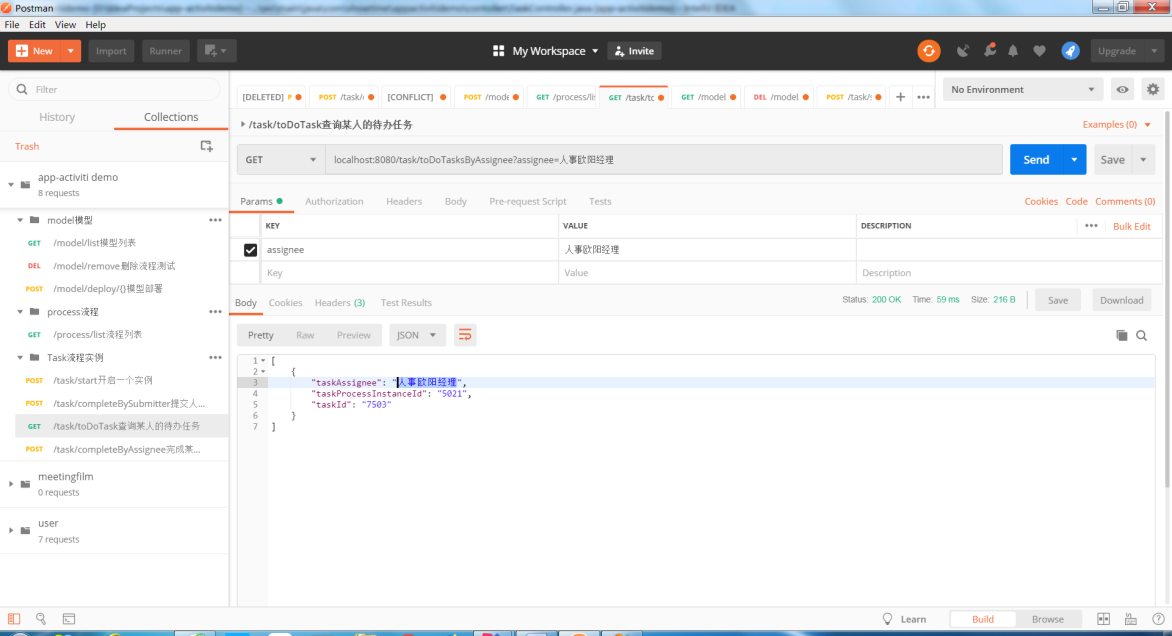


数据库数据有变动



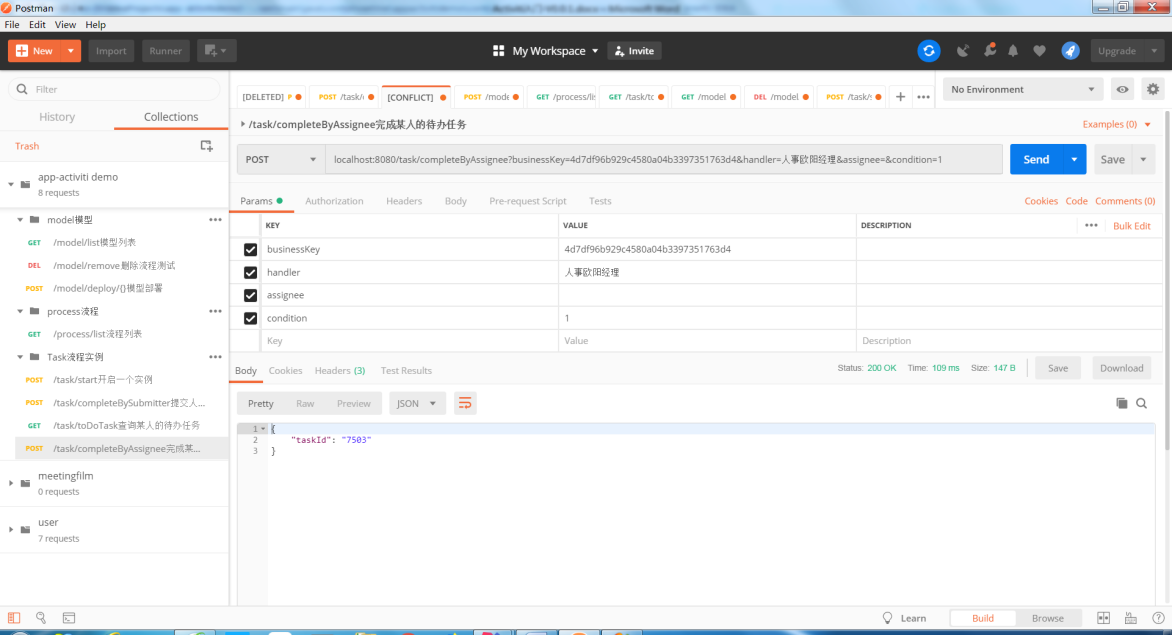
2.5.5人事查询待办任务：

localhost:8080/task/toDoTasksByAssignee?assignee=人事欧阳经理

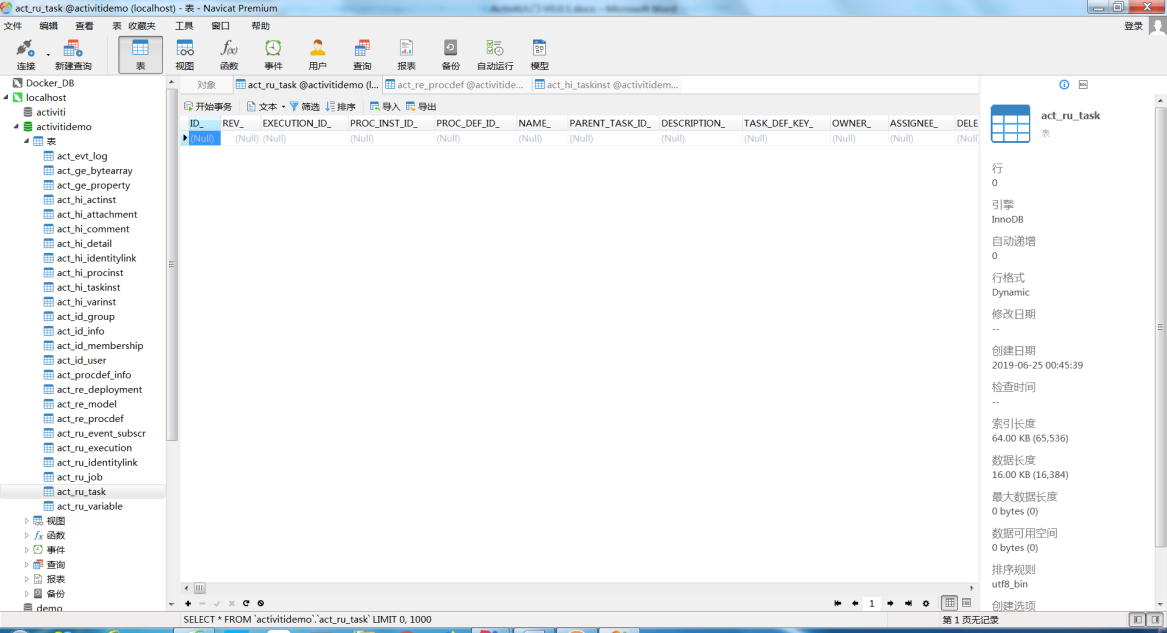


部门王经理通过审批，请假成功：

localhost:8080/task/completeByAssignee?businessKey=4d7df96b929c4580a04b3397351763d4&handler=人事欧阳经理&assignee=&condition=1

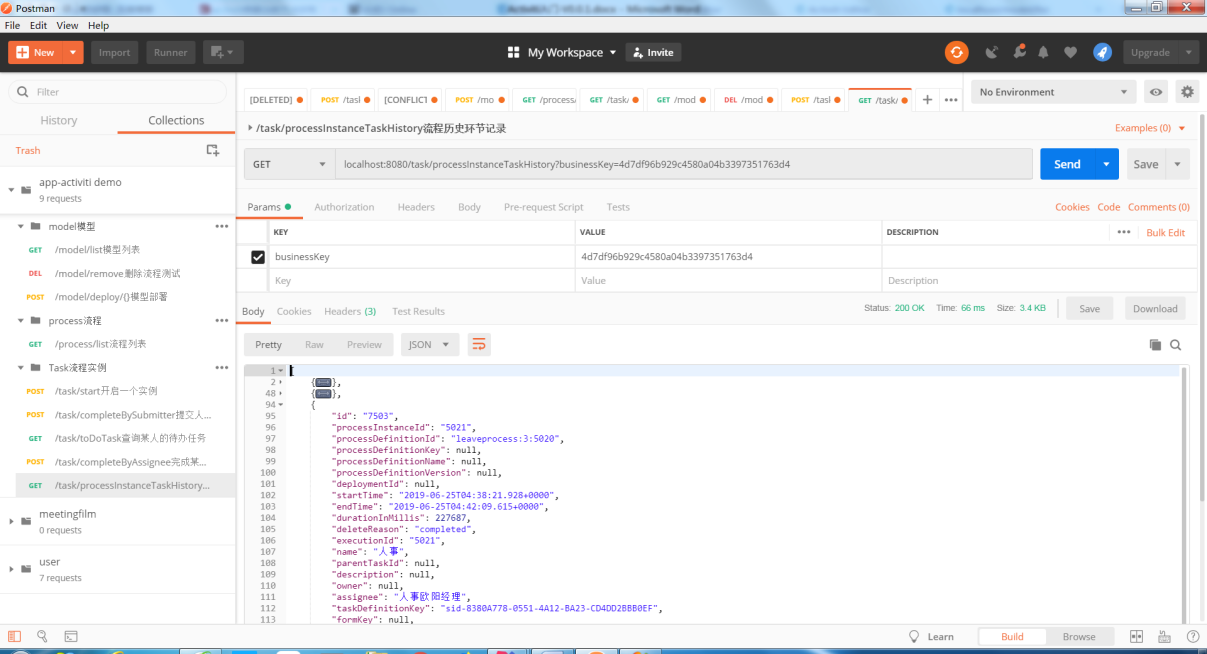


数据库数据被删除：



2.5.6 查看这个请假流程的历史记录

localhost:8080/task/processInstanceTaskHistory?businessKey=4d7df96b929c4580a04b3397351763d4



附项目地址：<https://github.com/zh666666/hello-world.git>

END

这个demo实现方法有多种，具体实现方式可以百度，加上自身理解实现。