

公告

昵称：颜色不一样的烟火
园龄： 10个月
粉丝： 6
关注： 1
[+加关注](#)

< 2020年1月 >						
日	一	二	三	四	五	六
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

搜索

找找看

谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

gitlab(4)

Activiti（二） springBoot2集成activiti，集成activiti在线设计器

. 摘要

本篇随笔主要记录springBoot2集成activiti流程引擎，并且嵌入activiti的在线设计器，可以通过浏览器直接编辑出我们需要的流程，不需要通过eclipse或者IDEA的actiBpm插件设计流程再直接部署到项目下，页面保存流程后可直接发布、发起流程。

. 所需软件版本

springBoot 2.0.1.RELEASE

activiti 5.22.0 [官网下载地址](#)

activiti-webapp-explorer2 [我的github上的explorer项目](#)

. 集成方法

activiti支持多种数据库，本身还有个h2的内存数据库，我这里用的是Mysql

1、pom.xml引入依赖，这里主要是activiti的依赖包，\${activiti.version}为5.22.0



```
<!--spring activiti start-->
    <dependency>
        <groupId>org.activiti</groupId>
        <artifactId>activiti-spring-boot-starter-basic</artifactId>
        <version>${activiti.version}</version>
        <exclusions>
```

jenkins(2)
activiti(2)
CentOs(2)
spring(2)
springboot(2)
Vmware(2)
动态代理(1)
设计模式(1)
项目构建(1)
更多

随笔分类

activiti(2)
centos(3)
gitlab(4)
jenkins(2)
js/jquery(1)
linux(4)
Mq(1)
mysql(1)
myBatis(1)
oracle(1)
redis(1)
solr(1)
spring(4)
springboot(3)
springCloud(1)

随笔档案

2019年12月(2)
2019年8月(1)
2019年7月(1)
2019年6月(2)
2019年5月(5)
2019年3月(2)
2019年2月(9)

最新评论

1. Re:Activiti （二） springBoot2集成activiti，集成activiti在线设计器
楼主可否给个联系方式，有问题请教
--沉迷睡觉的死宅
2. Re:Activiti （二） springBoot2集成activiti，集成activiti在线设计器

```
<exclusion>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</exclusion>
<exclusion>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</exclusion>
<exclusion>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
</exclusion>
</exclusions>
</dependency>
<dependency>
    <groupId>org.activiti</groupId>
    <artifactId>activiti-spring-boot-starter-actuator</artifactId>
    <version>${activiti.version}</version>
    <exclusions>
        <exclusion>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.activiti</groupId>
    <artifactId>activiti-rest</artifactId>
    <version>${activiti.version}</version>
    <exclusions>
        <exclusion>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.activiti</groupId>
    <artifactId>activiti-explorer</artifactId>
    <version>${activiti.version}</version>
    <exclusions>
        <exclusion>
            <groupId>com.vaadin</groupId>
            <artifactId>vaadin</artifactId>
        </exclusion>
        <exclusion>
            <groupId>org.vaadin.addons</groupId>
            <artifactId>dcharts-widget</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

楼主可否给个联系方式，有问题请教

--九惜

3. Re:Activiti（二）springBoot2集成activiti，集成activiti在线设计器
楼主，为什么我设置了javaBean作为变量的时候，获取的时候，返回的是ObjectNode类型，无法转换成JavaBean类型

--园子里的人

4. Re:Activiti（二）springBoot2集成activiti，集成activiti在线设计器
版主 开启流程的时候报这个no processes deployed with key 'modelKey'

--xiaoyuemushan

5. Re:Activiti（二）springBoot2集成activiti，集成activiti在线设计器
源码已经发到码云，各位可以去拉取
地址：

--颜色不一样的烟火

阅读排行榜

1. Activiti（二）springBoot2集成activiti，集成activiti在线设计器(6638)
2. springBoot添加日志管理(2779)
3. java实现word生成并转pdf(1585)
4. linux下oracle无法删除用户(618)
5. MYSQL SQL语句优化(547)

评论排行榜

1. Activiti（二）springBoot2集成activiti，集成activiti在线设计器(24)
2. java实现word生成并转pdf(2)

推荐排行榜

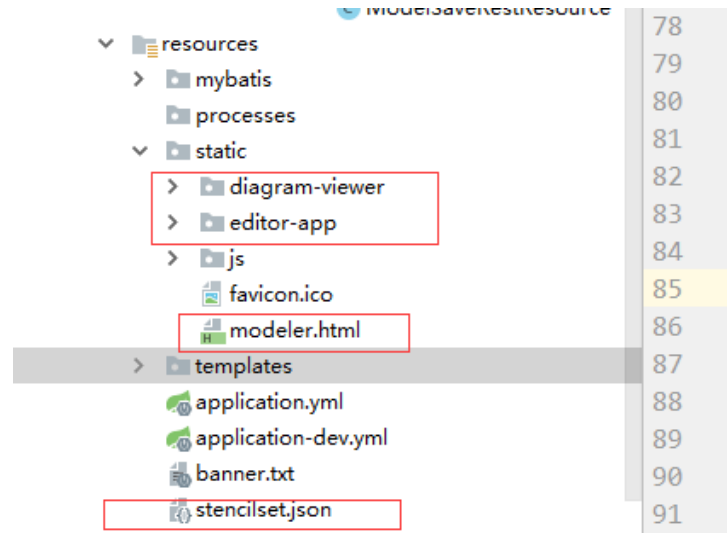
1. Activiti（二）springBoot2集成activiti，集成activiti在线设计器(1)
2. 数据库缓存mybatis,redis(1)
3. MYSQL SQL语句优化(1)
4. Linux环境oracle导库步骤(1)

```
<exclusion>
    <groupId>activiti-simple-workflow</groupId>
    <artifactId>org.activiti</artifactId>
</exclusion>
</exclusions>
</dependency>
<dependency>
    <groupId>org.activiti</groupId>
    <artifactId>activiti-diagram-rest</artifactId>
    <version>${activiti.version}</version>
</dependency>
<dependency>
    <groupId>org.activiti</groupId>
    <artifactId>activiti-simple-workflow</artifactId>
    <version>${activiti.version}</version>
</dependency>
<dependency>
    <groupId>org.activiti</groupId>
    <artifactId>activiti-spring</artifactId>
    <version>${activiti.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.xmlgraphics</groupId>
    <artifactId>batik-codec</artifactId>
    <version>1.7</version>
</dependency>
<dependency>
    <groupId>org.apache.xmlgraphics</groupId>
    <artifactId>batik-css</artifactId>
    <version>1.7</version>
</dependency>
<dependency>
    <groupId>org.apache.xmlgraphics</groupId>
    <artifactId>batik-svg-dom</artifactId>
    <version>1.7</version>
</dependency>
<dependency>
    <groupId>org.apache.xmlgraphics</groupId>
    <artifactId>batik-svggen</artifactId>
    <version>1.7</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>4.1.3.RELEASE</version>
</dependency>

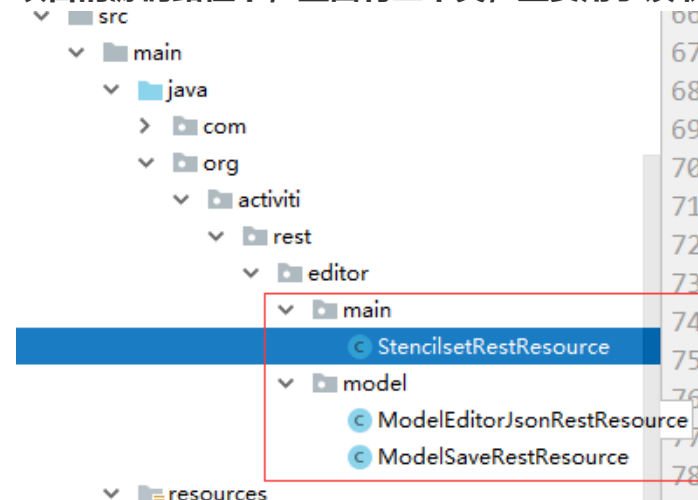
<!--spring activiti end-->
```

2、拷贝项目资源

解压activiti-webapp-explorer2，把webapp下面的diagram-viewer、editor-app、modeler.html复制到springboot项目下的static下，这是activiti的在线设计器，modeler.html就是设计的主界面，复制resources下stencilset.json到自己的resources下。



解压activiti-5.22.0的包，将libs下的activiti-modeler-5.22.0-sources.jar解压出来，把org\activiti\rest\editor路径下的main、model文件夹复制到springboot项目的源码路径下，里面有三个类，主要用于读取stencilset.json。



3、修改拷贝过来的资源文件

修改StencilsetRestResource.java、ModelEditorJsonRestResource.java

a、ModelSaveRestResource.java 3个controller，加上@RequestMapping("/service")

```
@RestController
@RequestMapping("/service")
public class StencilsetRestResource {

    @RequestMapping(value="/editor/stencilset", method = RequestMethod.GET, produces = "application/javascript")
    public @ResponseBody String getStencilset() {
        InputStream stencilsetStream = this.getClass().getClassLoader().getResourceAsStream("stencilset.js");
        try {
            return IOUtils.toString(stencilsetStream, encoding: "utf-8");
        } catch (Exception e) {
            throw new ActivitiException("Error while loading stencil set", e);
        }
    }
}
```

修改editor-app下的app-cfg.js，把contextRoot后面改成 /service，和controller里面加的requestMapping要一致

```
var ACTIVITI = ACTIVITI || {};
```

```
ACTIVITI.CONFIG = {
    'contextRoot' : '/service',
};
```

4、新增service、controller

service实现，这里只放实现类，接口就不放了，自己可以根据实现类自己添加接口方法，这个service实现了把流程和自定义表单结合的功能，completeTask方法通过java反射执行不同任务节点执行不同form



```

package com.djkj.activiti.service.impl;

import com.djkj.activiti.bean.ActivitiVariable;
import com.djkj.activiti.common.BusinessTaskUtil;
import com.djkj.activiti.service.ActivitiService;
import com.djkj.activiti.service.ActivitiVariableService;
import org.activiti.bpmn.model.BpmnModel;
import org.activiti.engine.HistoryService;
import org.activiti.engine.RepositoryService;
import org.activiti.engine.RuntimeService;
import org.activiti.engine.TaskService;
import org.activiti.engine.delegate.DelegateExecution;
import org.activiti.engine.history.HistoricActivityInstance;
import org.activiti.engine.history.HistoricProcessInstance;
import org.activiti.engine.impl.cfg.ProcessEngineConfigurationImpl;
import org.activiti.engine.impl.persistence.entity.ProcessDefinitionEntity;
import org.activiti.engine.impl.pvm.PvmTransition;
import org.activiti.engine.impl.pvm.process.ActivityImpl;
import org.activiti.engine.runtime.ProcessInstance;
import org.activiti.engine.task.Task;
import org.activiti.image.ProcessDiagramGenerator;
import org.apache.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import sun.misc.BASE64Encoder;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.lang.reflect.Method;
import java.util.*;

@Service
public class ActivitiServiceImpl implements ActivitiService {

    @Autowired
    private RuntimeService runtimeService;
    @Autowired
    private TaskService taskService;
    @Autowired
    private HistoryService historyService;
    @Autowired
    private RepositoryService repositoryService;
    @Autowired
    private ProcessEngineConfigurationImpl processEngineConfiguration;
    @Autowired

```

```

private ActivitiVariableService activitiVariableService;

private static final Logger logger=Logger.getLogger(ActivitiServiceImpl.class);

/**
 * 启动流程
 */
@Override
public void startProcesses(String id,String business_key) {
    ProcessInstance pi = runtimeService.startProcessInstanceByKey(id, business_key);
//流程图id, 业务表id
    System.out.println("流程启动成功, 流程id:"+pi.getId());
}

/**
 *
 * <p>描述: 根据用户id查询待办任务列表</p>
 * @author 范相如
 * @date 2018年2月25日
 */
@Override
public List<Task> findTasksByUserId(String userId) {
    List<Task> resultTask = taskService.createTaskQuery().processDefinitionKey("process").taskCandidateOrAssigned(userId).list();
    return resultTask;
}

@Override
public Task findTaskById(String taskId){
    List<Task> resultTask = taskService.createTaskQuery().taskId(taskId).list();
    if(resultTask!=null){
        return resultTask.get(0);
    }
    return null;
}

/**
 *
 * <p>描述:任务审批      (通过/拒接) </p>
 * @author 范相如
 * @date 2018年2月25日
 * @param taskId 任务id
 * @param userId 用户id
 * @param result false OR true
 */
@Override
public void completeTask(String taskId,String userId,String result) {
    //获取流程实例
    taskService.claim(taskId, userId);
}

```

```

        //获取任务
        Task task=taskService.createTaskQuery().taskId(taskId).singleResult();
        //获取流程实例ID
        String proInsId = task.getProcessInstanceId();
        //获取流程实例
        ProcessInstance process = runtimeService.createProcessInstanceQuery().processInst
anceId(proInsId).singleResult();
        //获取业务外键
        String business_key = process.getBusinessKey();
        String[] array = business_key.split(":");
        String business_Id = array[1];
        //业务处理
        try {
            Class clazz=BusinessTaskUtil.class;
            Object obj = clazz.newInstance();
            Method method = clazz.getMethod("actBusiness_"+task.getFormKey(),String.class
,String.class,String.class);
            method.invoke(obj,userId,business_Id,result);
        }catch (Exception e){
            e.printStackTrace();
            logger.error("执行业务方法错误! ");

        }
        taskService.complete(taskId);
    }

    /**
     * 更改业务流程状态#{ActivityDemoServiceImpl.updateBizStatus(execution,"tj")}
     * @param execution
     * @param status
     */
    @Override
    public void updateBizStatus(DelegateExecution execution, String status) {
        String bizId = execution.getProcessBusinessKey();
        //根据业务id自行处理业务表
        System.out.println("业务表["+bizId+"]状态更改成功, 状态更改为: "+status);
    }

    //流程节点权限用户列表#{ActivityDemoServiceImpl.findUsers(execution,sign)}
    @Override
    public List<String> findUsersForSL(DelegateExecution execution){
        return Arrays.asList("sly1","sly2");
    }

    //流程节点权限用户列表#{ActivityDemoServiceImpl.findUsers(execution,sign)}
    @Override
    public List<String> findUsersForSP(DelegateExecution execution){
        return Arrays.asList("spy1","uspy2");
    }

```



```

    }

    /**
     *
     * <p>描述： 生成流程图
     * 首先启动流程，获取processInstanceId，替换即可生成</p>
     * @author 范相如
     * @date 2018年2月25日
     * @param processInstanceId
     * @throws Exception
     */
    @Override
    public void queryProImg(String processInstanceId) throws Exception {
        //获取历史流程实例
        HistoricProcessInstance processInstance = historyService.createHistoricProcessInstanceQuery().processInstanceId(processInstanceId).singleResult();

        //根据流程定义获取输入流
        InputStream is = repositoryService.getProcessDiagram(processInstance.getProcessDefinitionId());
        BufferedImage bi = ImageIO.read(is);
        File file = new File("demo2.png");
        if(!file.exists()) file.createNewFile();
        FileOutputStream fos = new FileOutputStream(file);
        ImageIO.write(bi, "png", fos);
        fos.close();
        is.close();
        System.out.println("图片生成成功");

        List<Task> tasks = taskService.createTaskQuery().taskCandidateUser("userId").list();
        for(Task t : tasks) {
            System.out.println(t.getName());
        }
    }

    /**
     * 流程图高亮显示
     * 首先启动流程，获取processInstanceId，替换即可生成
     * @throws Exception
     */
    @Override
    public String queryProHighLighted(String processInstanceId) throws Exception {
        //获取历史流程实例
        HistoricProcessInstance processInstance = historyService.createHistoricProcessInstanceQuery().processInstanceId(processInstanceId).singleResult();
        //获取流程图
        BpmnModel bpmnModel = repositoryService.getBpmnModel(processInstance.getProcessDe

```

```

        finitionId());

        ProcessDiagramGenerator diagramGenerator = processEngineConfiguration.getProcessDiagramGenerator();

        ProcessDefinitionEntity definitionEntity = (ProcessDefinitionEntity)repositoryService.getProcessDefinition(processInstance.getProcessDefinitionId());

        List<HistoricActivityInstance> highLightedActivitList = historyService.createHistoricActivityInstanceQuery().processInstanceId(processInstanceId).list();
        //高亮环节id集合
        List<String> highLightedActivitis = new ArrayList<String>();

        //高亮线路id集合
        List<String> highLightedFlows = getHighLightedFlows(definitionEntity, highLightedActivitList);

        for(HistoricActivityInstance tempActivity : highLightedActivitList){
            String activityId = tempActivity.getActivityId();
            highLightedActivitis.add(activityId);
        }
        //配置字体
        InputStream imageStream = diagramGenerator.generateDiagram(bpmnModel, "png", highLightedActivitis, highLightedFlows, "宋体", "微软雅黑", "黑体", null, 2.0);
        BufferedImage bi = ImageIO.read(imageStream);
        //        File file = new File("demo2.png");
        //        if(!file.exists()) file.createNewFile();
        //        FileOutputStream fos = new FileOutputStream(file);
        ByteArrayOutputStream bos= new ByteArrayOutputStream();
        ImageIO.write(bi, "png", bos);
        byte[] bytes = bos.toByteArray();//转换成字节
        BASE64Encoder encoder = new BASE64Encoder();
        String png_base64 = encoder.encodeBuffer(bytes);//转换成base64串
        png_base64 = png_base64.replaceAll("\n", "").replaceAll("\r", ""); //删除 \r\n
        bos.close();
        imageStream.close();
        return png_base64;
    }
    /**
     * 获取需要高亮的线
     * @param processDefinitionEntity
     * @param historicActivityInstances
     * @return
     */
    private List<String> getHighLightedFlows(
        ProcessDefinitionEntity processDefinitionEntity,
        List<HistoricActivityInstance> historicActivityInstances) {

        List<String> highFlows = new ArrayList<String>();//用以保存高亮的线flowId
        for (int i = 0; i < historicActivityInstances.size() - 1; i++) { // 对历史流程节点进

```

行遍历

```
ActivityImpl activityImpl = processDefinitionEntity
    .findActivity(historicActivityInstances.get(i)
        .getActivityId()); // 得到节点定义的详细信息

List<ActivityImpl> sameStartTimeNodes = new ArrayList<ActivityImpl>(); // 用以
保存后需开始时间相同的节点

ActivityImpl sameActivityImpl1 = processDefinitionEntity
    .findActivity(historicActivityInstances.get(i + 1)
        .getActivityId());
// 将后面第一个节点放在时间相同节点的集合里
sameStartTimeNodes.add(sameActivityImpl1);
for (int j = i + 1; j < historicActivityInstances.size() - 1; j++) {
    HistoricActivityInstance activityImpl1 = historicActivityInstances
        .get(j); // 后续第一个节点
    HistoricActivityInstance activityImpl2 = historicActivityInstances
        .get(j + 1); // 后续第二个节点
    if (activityImpl1.getStartTime().equals(
        activityImpl2.getStartTime())) {
        // 如果第一个节点和第二个节点开始时间相同保存
        ActivityImpl sameActivityImpl2 = processDefinitionEntity
            .findActivity(activityImpl2.getActivityId());
        sameStartTimeNodes.add(sameActivityImpl2);
    } else {
        // 有不相同跳出循环
        break;
    }
}

List<PvmTransition> pvmTransitions = activityImpl
    .getOutgoingTransitions(); // 取出节点的所有出去的线
for (PvmTransition pvmTransition : pvmTransitions) {
    // 对所有的线进行遍历
    ActivityImpl pvmActivityImpl = (ActivityImpl) pvmTransition
        .getDestination();
    // 如果取出的线的目标节点存在时间相同的节点里，保存该线的id，进行高亮显示
    if (sameStartTimeNodes.contains(pvmActivityImpl)) {
        highFlows.add(pvmTransition.getId());
    }
}

}

return highFlows;
}
}
```



controller实现，我写了两个controller，主要当时写的时候没有写在一起，你们可以把这两部分的内容合并在一起



```
package com.djkj.activiti.controller;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ObjectNode;
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.ProcessEngine;
import org.activiti.engine.RepositoryService;
import org.activiti.engine.repository.Deployment;
import org.activiti.engine.repository.Model;
import org.apache.commons.lang3.StringUtils;
import org.apache.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.util.List;

@Controller
@RequestMapping("/models")
public class MyActivitiController {

    private static final Logger LOGGER = Logger.getLogger(MyActivitiController.class);

    @Autowired
    private RepositoryService repositoryService;
    @Autowired
    private ObjectMapper objectMapper;

    @RequestMapping("/modelList")
    public String modelList(org.springframework.ui.Model model,HttpServletRequest request
```

```

t){
    LOGGER.info("-----列表页-----");
    List<Model> models = repositoryService.createModelQuery().orderByCreateTime().desc().list();
    model.addAttribute("models",models);
    String info = request.getParameter("info");
    if(StringUtils.isEmpty(info)){
        model.addAttribute("info",info);
    }
    return "model/list";
}

@RequestMapping("/create")
public void newModel(HttpServletRequest request, HttpServletResponse response) throws
UnsupportedEncodingException {
    try {
        //初始化一个空模型
        Model model = repositoryService.newModel();

        //设置一些默认信息
        String name = "new-process";
        String description = "";
        int revision = 1;
        String key = "process";

        ObjectNode modelNode = objectMapper.createObjectNode();
        modelNode.put(ModelDataJsonConstants.MODEL_NAME, name);
        modelNode.put(ModelDataJsonConstants.MODEL_DESCRIPTION, description);
        modelNode.put(ModelDataJsonConstants.MODEL_REVISION, revision);

        model.setName(name);
        model.setKey(key);
        model.setMetaInfo(modelNode.toString());

        repositoryService.saveModel(model);
        String id = model.getId();

        //完善ModelEditorSource
        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);
        repositoryService.addModelEditorSource(id, editorNode.toString().getBytes("utf-8"));

        response.sendRedirect(request.getContextPath() + "/static/modeler.html?modelId

```

```

    =" + id);
    } catch (IOException e) {
        e.printStackTrace();
        LOGGER.info("模型创建失败! ");
    }

}

@RequestMapping("/delete/{id}")
public @ResponseBody String deleteModel(@PathVariable("id")String id){
    repositoryService.deleteModel(id);
    return "删除成功! ";
}

@RequestMapping("/deployment/{id}")
public @ResponseBody String deploy(@PathVariable("id")String id) throws Exception {

    //获取模型
    Model modelData = repositoryService.getModel(id);
    byte[] bytes = repositoryService.getModelEditorSource(modelData.getId());

    if (bytes == null) {
        return "模型数据为空, 请先设计流程并成功保存, 再进行发布。";
    }

    JsonNode modelNode = new ObjectMapper().readTree(bytes);

    BpmnModel model = new BpmnJsonConverter().convertToBpmnModel(modelNode);
    if(model.getProcesses().size()==0){
        return "数据模型不符合要求, 请至少设计一条主线流程。";
    }
    byte[] bpmnBytes = new BpmnXMLConverter().convertToXML(model);

    //发布流程
    String processName = modelData.getName() + ".bpmn20.xml";
    Deployment deployment = repositoryService.createDeployment()
        .name(modelData.getName())
        .addString(processName, new String(bpmnBytes, "UTF-8"))
        .deploy();
    modelData.setDeploymentId(deployment.getId());
    repositoryService.saveModel(modelData);
    return "流程发布成功";
}

@RequestMapping("/start/{id}")
public @ResponseBody String startProcess(@PathVariable("id") String id){
    try {

```

```

        }catch (Exception e){
            e.printStackTrace();
            return "流程启动失败! ";
        }
        return "流程启动成功";
    }
}

```



显示所有流程列表，以及流程的增删改



```

package com.djkj.activiti.controller;

import com.djkj.activiti.bean.ActivitiVariable;
import com.djkj.activiti.bean.FormData;
import com.djkj.activiti.service.ActivitiService;
import com.djkj.activiti.service.ActivitiVariableService;
import org.activiti.engine.HistoryService;
import org.activiti.engine.TaskService;
import org.activiti.engine.task.Task;
import org.apache.commons.lang3.StringUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.servlet.mvc.support.RedirectAttributesModelMap;

import javax.annotation.Resource;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@Controller
@RequestMapping("/activiti")
public class ActivitiProcessController {
    @Resource
    private ActivitiService activitiServiceImpl;
}

```

```

@Autowired
private TaskService taskService;
@Autowired
private HistoryService historyService;
@Autowired
private ActivitiVariableService activitiVariableService;

// private static final String business = "testtask";

//启动流程---215001
@RequestMapping("/start/{key}")
public String startProcess(@PathVariable("key") String key, RedirectAttributes modelMap) {
    // Map<String,Object> map = new HashMap<String,Object>();
    // map.put("userId",userIdSl);
    // map.put("businessKey",business_key);
    // String business_key = key + ":" + key;
    ActivitiVariable activitiVariable = new ActivitiVariable();
    int resul=activitiVariableService.insert(activitiVariable);
    if(resul==1){
        modelMap.addFlashAttribute("info","流程启动成功! ");
        String business_key = key+":"+activitiVariable.getId();
        activitiServiceImpl.startProcesses(key,business_key);
    }else{
        modelMap.addFlashAttribute("info","流程启动失败! ");
    }
    return "redirect:/models/modelList";
}

//获取受理员任务列表
@RequestMapping("/queryTaskSl")
public String findTasksForSL(ModelMap modelMap, FormData formData) {
    List<Task> lists = activitiServiceImpl.findTasksByUserId(formData.getUserId());
    modelMap.addAttribute("tasks",lists);
    modelMap.addAttribute("userId",formData.getUserId());
    return "model/taskList";
}

@RequestMapping("/form")
public String form(FormData formData, ModelMap modelMap) {
    Task task=activitiServiceImpl.findTaskById(formData.getId());
    modelMap.addAttribute("data",formData);
    modelMap.addAttribute("task",task);
    if(StringUtils.isEmpty(task.getFormKey())){
        return "activitiForm/"+task.getFormKey();
    }
    return "model/form";
}
}

```



```

//受理员受理数据
@RequestMapping("/completeTaskSl")
public String completeTasksForSL(ModelMap modelMap, FormData formData) {
    activitiServiceImpl.completeTask(formData.getId(), formData.getUserId(), formData
a.getAttr1()); //受理后, 任务列表数据减少
    return findTasksForSL(modelMap, formData);
}

//    //获取审批员任务列表
//    @RequestMapping("/queryTaskSp")
//    public void findTasksForSP() {
//        List<Task> lists = activitiServiceImpl.findTasksByUserId(userIdSp);
//        System.out.println("任务列表: "+lists); //任务列表: [Task[id=220004, name=审批]]
//    }
//
//    //审批员通过审核
//    @RequestMapping("/completeTaskSp/{id}")
//    public void completeTasksForSP(@PathVariable("id") String id) {
//        activitiServiceImpl.completeTask(id, userIdSp, "true"); //审批后, 任务列表数据减少
//    }

//设置流程变量
//设置流程变量【基本类型】
public void setTasksVar() {
    List<Task> lists = activitiServiceImpl.findTasksByUserId("sly1");
    for(Task task : lists) { //不知为何, 变量保存成功, 但数据表只有请假天数含有任务id, 单获取流程
变量时, 根据任务id均可获取到 (如下一测试)
        taskService.setVariable(task.getId(), "请假人", "sly1");
        taskService.setVariableLocal(task.getId(), "请假天数", 3);
        taskService.setVariable(task.getId(), "请假日期", new Date());
    }
}

//    //获取流程变量
//    @Test
//    public void getTasksVar() {
//        List<Task> lists = activitiServiceImpl.findTasksByUserId("sly1");
//        for(Task task : lists) {
//            //获取流程变量【基本类型】
//            String person = (String) taskService.getVariable(task.getId(), "请假人");
//            Integer day = (Integer) taskService.getVariableLocal(task.getId(), "请假天
数");
//            Date date = (Date) taskService.getVariable(task.getId(), "请假日期");
//
//            System.out.println("流程变量: "+person+"||"+day+"||"+date+"||");
//        }

```

```

//    }
//
//    //设置流程变量【实体】
//    @Test
//    public void setTasksVarEntity() {
//        List<Task> lists = activitiServiceImpl.findTasksByUserId("sly1");
//        for(Task task : lists) {
//            Person p = new Person();
//            p.setName("翠花");
//            p.setId(20);
//            p.setDate();
//            p.setNote("回去探亲, 一起吃饭123");
//            taskService.setVariable(task.getId(), "人员信息(添加固定版本)", p);
//
//            System.out.println("设置流程变量成功! ");
//        }
//    }
//
//    //获取流程变量【实体】  实体必须序列化
//    @Test
//    public void getTasksVarEntity() {
//        List<Task> lists = activitiServiceImpl.findTasksByUserId("sly1");
//        for(Task task : lists) {
//            // 2.获取流程变量, 使用javaBean类型
//            Person p = (Person)taskService.getVariable(task.getId(), "人员信息(添加固定版本)");
//            System.out.println(" 请假人:  "+p.getName()+"  请假天数:  "+p.getId()+"  请
//            假时间: "+ p.getDate()+ "  请假原因:  "+p.getNote());
//        }
//    }
//
//
//    //生成流程图---232501
//    @RequestMapping("/queryProImg")
//    public void queryProImg() throws Exception {
//        activitiServiceImpl.queryProImg("232501");
//    }
//
//    //生成流程图 (高亮) ---232501
//    @RequestMapping("/queryProHighLighted")
//    public @ResponseBody String queryProHighLighted(String proInsId) throws Exception {
//        String imageByteArray = activitiServiceImpl.queryProHighLighted(proInsId);
//        return imageByteArray;
//    }
//
//    /**
//     * 查询流程变量的历史表, 可以根据变量名称查询该变量的所有历史信息
//     */
//    @Test

```

```

//      public void findHistoryProcessVariables() {
//          List<HistoricVariableInstance> list = historyService.createHistoricVariableInst
anceQuery() //创建一个历史的流程变量查询对象
//              .variableName("请假天数")
//              .list();
//          if (list!=null &&list.size()>0) {
//              for (HistoricVariableInstance hvi : list) {
//                  System.out.println(hvi.getId()+"          "+hvi.getProcessInstanceId()+"
"+hvi.getVariableName()
//                      +""+hvi.getVariableTypeName()+"          "+hvi.getValue());
//                  System.out.println("#####");
//              }
//          }
//      }
//  }
//  /**
//   * 历史流程实例查询
//   * http://blog.csdn.net/luckyzhoustar/article/details/48652783
//   */
//  @Test
//  public void findHistoricProcessInstance() {
//      // 查询已完成的流程
//      List<HistoricProcessInstance> datas = historyService
//          .createHistoricProcessInstanceQuery().finished().list();
//      System.out.println("使用finished方法: " + datas.size());
//      // 根据流程定义ID查询
//      datas = historyService.createHistoricProcessInstanceQuery()
//          .processDefinitionId("processDefinitionId").list();
//      System.out.println("使用processDefinitionId方法: " + datas.size());
//      // 根据流程定义key (流程描述文件的process节点id属性) 查询
//      datas = historyService.createHistoricProcessInstanceQuery()
//          .processDefinitionKey("processDefinitionKey").list();
//      System.out.println("使用processDefinitionKey方法: " + datas.size());
//      // 根据业务主键查询
//      datas = historyService.createHistoricProcessInstanceQuery()
//          .processInstanceBusinessKey("processInstanceBusinessKey").list();
//      System.out.println("使用processInstanceBusinessKey方法: " + datas.size());
//      // 根据流程实例ID查询
//      datas = historyService.createHistoricProcessInstanceQuery()
//          .processInstanceId("processInstanceId").list();
//      System.out.println("使用processInstanceId方法: " + datas.size());
//      // 查询没有完成的流程实例
//      historyService.createHistoricProcessInstanceQuery().unfinished().list();
//      System.out.println("使用unfinished方法: " + datas.size());
//  }
//  /**

```

```
// * 历史任务查询
// * @throws ParseException
// */
// @Test
// public void findHistoricTasks() throws ParseException {
//     SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
//
//     //历史数据查询
//     List<HistoricTaskInstance> datas = historyService.createHistoricTaskInstanceQue
ry()
//         .finished().list();
//     System.out.println("使用finished方法查询: " + datas.size());
//     datas = historyService.createHistoricTaskInstanceQuery()
//         .processDefinitionId("processDefinitionId").list();
//     System.out.println("使用processDefinitionId方法查询: " + datas.size());
//     datas = historyService.createHistoricTaskInstanceQuery()
//         .processDefinitionKey("testProcess").list();
//     System.out.println("使用processDefinitionKey方法查询: " + datas.size());
//     datas = historyService.createHistoricTaskInstanceQuery()
//         .processDefinitionName("testProcess2").list();
//     System.out.println("使用processDefinitionName方法查询: " + datas.size());
//     datas = historyService.createHistoricTaskInstanceQuery()
//         .processFinished().list();
//     System.out.println("使用processFinished方法查询: " + datas.size());
//     datas = historyService.createHistoricTaskInstanceQuery()
//         .processInstanceId("processInstanceId").list();
//     System.out.println("使用processInstanceId方法查询: " + datas.size());
//     datas = historyService.createHistoricTaskInstanceQuery()
//         .processUnfinished().list();
//     System.out.println("使用processUnfinished方法查询: " + datas.size());
//     datas = historyService.createHistoricTaskInstanceQuery()
//         .taskAssignee("crazyit").list();
//     System.out.println("使用taskAssignee方法查询: " + datas.size());
//     datas = historyService.createHistoricTaskInstanceQuery()
//         .taskAssigneeLike("%zy%").list();
//     System.out.println("使用taskAssigneeLike方法查询: " + datas.size());
//     datas = historyService.createHistoricTaskInstanceQuery()
//         .taskDefinitionKey("usertask1").list();
//     System.out.println("使用taskDefinitionKey方法查询: " + datas.size());
//     datas = historyService.createHistoricTaskInstanceQuery()
//         .taskDueAfter(sdf.parse("2020-10-11 06:00:00")).list();
//     System.out.println("使用taskDueAfter方法查询: " + datas.size());
//     datas = historyService.createHistoricTaskInstanceQuery()
//         .taskDueBefore(sdf.parse("2022-10-11 06:00:00")).list();
//     System.out.println("使用taskDueBefore方法查询: " + datas.size());
//     datas = historyService.createHistoricTaskInstanceQuery()
//         .taskDueDate(sdf.parse("2020-10-11 06:00:00")).list();
//     System.out.println("使用taskDueDate方法查询: " + datas.size());
//     datas = historyService.createHistoricTaskInstanceQuery()
```

```

//      .unfinished().list();
//      System.out.println("使用unfinished方法查询: " + datas.size());
//  }
//  /**
//   *   历史行为查询
//   *   流程在进行过程中, 每每走一个节点, 都会记录流程节点的信息, 包括节点的id, 名称、类型、时间等, 保
//   存到ACT_HI_ACTINST表中。
//   * @throws ParseException
//   */
//   @Test
//   public void findHistoricActivityInstance() {
//       SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
//
//       //查询数据
//       List<HistoricActivityInstance> datas = historyService.createHistoricActivityIns
// tanceQuery()
//           .activityId("endevent1").list();
//       System.out.println("使用activityId查询: " + datas.size());
//       datas = historyService.createHistoricActivityInstanceQuery()
//           .activityInstanceId(datas.get(0).getId()).list();
//       System.out.println("使用activityInstanceId查询: " + datas.size());
//       datas = historyService.createHistoricActivityInstanceQuery()
//           .activityType("intermediateSignalCatch").list();
//       System.out.println("使用activityType查询: " + datas.size());
//       datas = historyService.createHistoricActivityInstanceQuery()
//           .executionId("executionId").list();
//       System.out.println("使用executionId查询: " + datas.size());
//       datas = historyService.createHistoricActivityInstanceQuery().finished().list();
//       System.out.println("使用finished查询: " + datas.size());
//       datas = historyService.createHistoricActivityInstanceQuery()
//           .processInstanceId("processInstanceId").list();
//       System.out.println("使用processInstanceId查询: " + datas.size());
//       datas = historyService.createHistoricActivityInstanceQuery()
//           .taskAssignee("crazyit").list();
//       System.out.println("使用taskAssignee查询: " + datas.size());
//       datas = historyService.createHistoricActivityInstanceQuery().unfinished().list
// ();
//       System.out.println("使用unfinished查询: " + datas.size());
//   }
//   /**
//   *   历史流程明细查询
//   *   在流程进行的过程中, 会产生许多明细数据, 只有将History设置为最高级别的时候, 才会被记录到ACT_HI
//   _DETAIL表中。
//   * @throws ParseException
//   */
//   @Test
//   public void findHistoricDetail() {
//       // 查询历史行为

```

```
// HistoricActivityInstance act = historyService.createHistoricActivityInstanceQue
ry()
//
//         .activityName("First Task").finished().singleResult();
//
// List<HistoricDetail> datas = historyService.createHistoricDetailQuery()
//
//         .activityInstanceId(act.getId()).list();
//
// System.out.println("使用activityInstanceId方法查询: " + datas.size());
//
// datas = historyService.createHistoricDetailQuery().excludeTaskDetails().list();
//
// System.out.println("使用excludeTaskDetails方法查询: " + datas.size());
//
// datas = historyService.createHistoricDetailQuery().formProperties().list();
//
// System.out.println("使用formProperties方法查询: " + datas.size());
//
// datas = historyService.createHistoricDetailQuery().processInstanceId("processIn
stanceId").list();
//
// System.out.println("使用processInstanceId方法查询: " + datas.size());
//
// datas = historyService.createHistoricDetailQuery().taskId("taskId").list();
//
// System.out.println("使用taskId方法查询: " + datas.size());
//
// datas = historyService.createHistoricDetailQuery().variableUpdates().list();
//
// System.out.println("使用variableUpdates方法查询: " + datas.size());
//
//     }
}

```

启动流程、用户任务task查询、业务处理、流程进度查看等等，历史流程查看功能没有验证过，这部分代码是注释掉的

5、页面实现

在templates放置显示页面，可自己根据喜好放置，我这里主要是跟controller里返回的路径对应

▼ templates	103
▼ activitiForm	104
applyForm.html	105
approveForm.html	106
▼ model	107
form.html	108
list.html	109
taskList.html	110

model文件夹下放的是模型列表页和用户任务列表页，form.html已经废弃，改为上面的activitiForm里面的form页面，不同的页面对应不同流程节点

list.html模型列表页



```
<!DOCTYPE html>
<html lang="zh" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script src="/static/js/jquery.min.js"></script>
    <script th:inline="javascript">
        function deployment(obj) {
            var id=obj.attributes['objectid'].nodeValue;
            $.ajax({
                url:"/models/deployment/"+id,
                type:"GET",
                success:function(res) {
                    alert(res);
                }
            });
        }

        function deleteProcess(obj) {
            var id=obj.attributes['objectid'].nodeValue;
            $.ajax({
                url:"/models/delete/"+id,
                type:"GET",
                success:function(res) {
                    alert(res);
                }
            });
        }

        function start(obj) {
            var id=obj.attributes['objectid'].nodeValue;
            $.ajax({
                url:'/models/start/'+id,
                type:'GET',
                success:function(res) {
                    alert(res);
                }
            });
        }
    </script>
</head>
</html>
```

```

    }

    function taskList() {
        $("#taskList").submit();
    }
</script>
</head>
<body>
<div class="info"><span th:text="${info}"></span></div>
<a href="/models/create">新建</a>
<form id="taskList" action="/activiti/queryTaskSl">
<div>
    <div style="float:left;width:50%;">
        <select id="userId" name="userId">
            <option value="userzhou">userzhou</option>
            <option value="usertest">usertest</option>
        </select>
    </div>
    <div style="width:50%;">
        <button onclick="taskList();">任务列表</button>
    </div>
</div>
</form>

<table class="table">
    <thead>
        <tr>
            <th>ID</th>
            <th>模型名称</th>
            <th>key</th>
            <th>版本</th>
            <th>部署ID</th>
            <th>创建时间</th>
            <th>最后更新时间</th>
            <th>操作</th>
        </tr>
    </thead>
    <tbody>
        <tr th:each="data : ${models}">
            <td th:text="${data.id}"></td>
            <td><a th:href="@{/static/modeler.html(modelId=${data.id})}" class="font-blue" th:text="${data.name}"></a>
            </td>
            <td th:text="${data.key}"></td>
            <td th:text="${data.version}"></td>
            <td th:text="${data.deploymentId}"></td>
            <td th:text="${data.createTime}"> 2018-02-25 17:28:35</td>

```



```
<td th:text="${data.lastUpdateTime}"> 2018-02-25 17:28:35</td>
<td><a href="javascript:void(0);" onclick="deployment(this);" th:attrappend="objectId=${data.id}" lass="font-blue deployBtn">发布</a>
<a th:href="'/activiti/start/'+data.key" th:attrappend="objectId=${data.id}" lass="font-blue deployBtn">发起流程</a>
<a th:href="@{/crm/model/export/{modelId} (modelId=${data.id})}" class="font-blue">导出</a>
<a href="javascript:void(0);" onclick="deleteProcess(this);" th:attrappend="objectId=${data.id}" class="font-blue delBtn">删除</a></td>
</tr>
</tbody>
</table>
</body>
</html>
```



taskList.html任务列表页



```
<!DOCTYPE html>
<html lang="zh" xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Title</title>
<script src="/static/js/jquery.min.js"></script>
</head>
<body>
<a href="/models/modelList"><button>返回模型列表页</button></a>
<div>
<span th:text="'用户: '+${userId}"></span>
<table>
<thead>
<tr>
<th>ID</th>
<th>任务名称</th>
<th>受理人</th>
<th>任务ID</th>
<th>流程ID</th>
<th>创建时间</th>
<th>操作</th>
</tr>
</thead>
<tbody>
<tr th:each="data : ${tasks}">
```

```

        <td th:text="${data.id}">LeaveBill:1:4</td>
        <td th:text="${data.name}"></td>
        <td th:text="${data.assignee}">1</td>
        <td th:text="${data.taskDefinitionKey}"></td>
        <td th:text="${data.processDefinitionId}"></td>
        <td th:text="${data.createTime}"> 2018-02-25 17:28:35</td>
        <td>
            <a th:href="${'/activiti/form?id='+data.id+'&userId='+data.assignee}" th:
attrappend="objectId=${data.id}" lass="font-blue deployBtn">处理</a>
        </td>
    </tr>
</tbody>
</table>
</div>
</body>
</html>

```



applyForm.html任务列表页



```

<!DOCTYPE html>
<html lang="zh" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>申请页面</title>
    <script src="/static/js/jquery.min.js"></script>
    <style>
        .title{
            text-align: center;
        }
    </style>
    <script>
        function showProcessImg(processInstanceId) {
            $.ajax({
                url:'/activiti/queryProHighLighted',
                type:'POST',
                data:{
                    proInsId:processInstanceId
                },
                success:function(res){
                    $("#processImg").attr('src',"data:image/png;base64,"+res);
                }
            })
        }
    </script>

```

```
}
function returnBack(userId) {
    var form = $("<form></form>");
    form.attr("action", "/activiti/queryTaskS1");
    form.attr("method", "post");

    var input = $("<input type='text' name='userId' />")
    input.attr("value", userId);
    form.append(input);

    form.appendTo("body");


    form.css("display", "none");

    form.submit();

}
</script>
</head>
<body>
<div class="title">申请表单</div>
<a th:onclick="'javascript:returnBack('${data.userId}');" "><button>返回用户任务列表</b
utton></a>
<a href="javascript:void(0);" th:onclick="'javascript:showProcessImg('${task.processIn
stanceId}+'\\')'" "><button>查看流程图</button></a>
<form action="/activiti/completeTaskS1">
    <input type="hidden" id="id" name="id" th:value="${data.id}"/>
    <input type="hidden" id="userId" name="userId" th:value="${data.userId}"/>
    <input type="text" id="attr1" name="attr1"/>
    <button type="submit">提交</button>
    <button>取消</button>
</form>
<img src="" id="processImg"/>
</body>
</html>
```



approveForm.html任务列表页

```
☐

<!DOCTYPE html>
<html lang="zh" xmlns:th="http://www.thymeleaf.org">
<head>
```

```
<meta charset="UTF-8">
<title>审批页面</title>
<script src="/static/js/jquery.min.js"></script>
<style>
    .title{
        text-align: center;
    }
</style>
<script>
    function showProcessImg(processInstanceId) {
        $.ajax({
            url: '/activiti/queryProHighLighted',
            type: 'POST',
            data: {
                proInsId: processInstanceId
            },
            success: function (res) {
                $("#processImg").attr('src', "data:image/png;base64," + res);
            }
        })
    }

    function returnBack(userId) {
        var form = $("<form></form>");
        form.attr("action", "/activiti/queryTaskSl");
        form.attr("method", "post");

        var input = $("<input type='text' name='userId' />");
        input.attr("value", userId);
        form.append(input);

        form.appendTo("body");

        form.css("display", "none");

        form.submit();

    }
</script>
</head>
<body>
<div class="title">审批表单</div>
<a th:onclick="'javascript:returnBack('${data.userId}');" "><button>返回任务列表</button></a>
<a href="javascript:void(0);" th:onclick="'javascript:showProcessImg('${task.processInstanceId}');" "><button>查看流程图</button></a>
<form action="/activiti/completeTaskSl">
    <input type="hidden" id="id" name="id" th:value="${data.id}"/>
    <input type="hidden" id="userId" name="userId" th:value="${data.userId}"/>
```

```
<input type="text" id="attr1" name="attr1"/>
<button type="submit">提交</button>
<button>取消</button>
</form>
<img src="" id="processImg"/>
</body>
</html>
```



运行界面如下

list.html列表页

新建

userzhou

任务列表

ID	模型名称	key	版本	部署ID	创建时间	最后更新时间	操作
1	MyActiviti-release	process 1	4		Tue Feb 19 16:43:30 CST 2019	Tue Feb 19 16:45:38 CST 2019	发布 发起流程 导出 删除

新建页面

Alfresco Activiti

Start Events

Activities

Structural

Gateways

Boundary Events

Intermediate Catching Events

Intermediate Throwing Events

End Events

Swimlanes

Artifacts

<

new-process

Process identifier :

process

Documentation :

No value

Process version string (documentation only) :

No value

Event listeners :

No event listeners configured

Message definitions :

No message definitions configured

Name :

No value

Process author :

No value

Target namespace :

http://www.activiti. ...

Execution listeners :

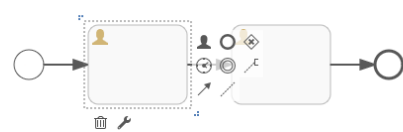
No execution listeners configured

Signal definitions :

No signal definitions configured

这里主要注意连个地方， assignments表示当前节点的处理人的id， 我自己定义了两个用户userzhou和usertest， form key填写处理当前节点所用的form表单， 也就是之前的applyForm和approveForm

Alfresco Activiti



▼ Start Events

- Start event
- ⌚ Start timer event
- Ⓐ Start signal event
- ✉ Start message event
- ⚠ Start error event

► Activities

► Structural

► Gateways

► Boundary Events

► Intermediate Catching Events

► Intermediate Throwing Events

► End Events

► Swimlanes

► Artifacts

< new-process

Exclusive :	<input type="checkbox"/>	Execution listeners :	No execution listeners configured
Multi-instance type :	None	Cardinality (Multi-instance) :	No value
Collection (Multi-instance) :	No value	Element variable (Multi-instance) :	No value
Completion condition (Multi-instance) :	No value	Is for compensation :	<input type="checkbox"/>
Assignments :	No assignment selected	Form key :	No value
Due date :	No value	Priority :	No value
Form properties :	No form properties selected	Task listeners :	No task listeners configured

Assignment



Assignee

userzhou

Candidate users

- +

Candidate groups

- +

Cancel

Save

[返回模型列表页](#)

用户：usertest

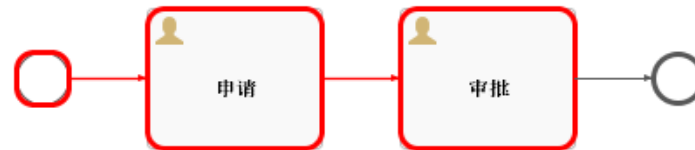
ID	任务名称	受理人	任务ID	流程ID	创建时间	操
10008	审批	usertest	sid-6CF35F45-3C08-4DBC-A02A-540553A0F69B	process:1:7	Tue Feb 19 17:08:36 CST 2019	处
12507	审批	usertest	sid-6CF35F45-3C08-4DBC-A02A-540553A0F69B	process:1:7	Tue Feb 19 17:10:28 CST 2019	处

返回任务列表

查看流程图

提交

取消



• 需要注意的地方

1、去除activiti的security校验，修改application启动类

```
@SpringBootApplication(scanBasePackages = {"com.djkj","org.activiti"} , exclude = { org.springframework.boot.autoconfigure.security.servlet.SecurityAutoConfiguration.class, org.activiti.spring.boot.SecurityAutoConfiguration.class,})
```

这是按照网上的写的，但还是存在校验，最后没办法在act_id_user里面加了一条数据，大家有解决的办法可以在下方给我评论，谢谢。

2、访问静态资源的时候，可能会被dispatcherServlet 给拦截掉，我是添加了个资源处理器，拦截static静态资源

```
package com.djkj.activiti.common;

import groovy.util.logging.Slf4j;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
```



```

import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
@Slf4j
@EnableWebMvc
public class StaticResourceConfig implements WebMvcConfigurer {

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/static/**").addResourceLocations("classpath:/static/");
    }

}

```

3、关闭编辑器的时候可以自定义返回的页面

修改editor-app\configuration下的toolbar-default-actions.js，
有俩个地方closeEditor方法和\$scope.saveAndClose方法，一个是关闭一个是保存并关闭，可以修改放回路径，对应的是controller里的路径

```

closeEditor: function (services) {
    window.location.href = "/models/modelList";
},

$scope.saveAndClose = function () {
    $scope.save(function () {
        window.location.href = "/models/modelList";
    });
};

```

分类: [activiti](#), [spring](#), [springboot](#)

标签: [springboot](#), [activiti](#)

好文要顶

关注我

收藏该文

